Integration and Communication of Engineering Information in Collaborative Design

Haoxue Ma

Luleå University of Technology
Department of Applied Physics and Mechanical Engineering, Division of Computer Aided Design
Integration and Communication of Engineering Information in Collaborative Design

Haoxue Ma

Division of Computer Aided Design
Department of Applied Physics and Mechanical Engineering
Luleå University of Technology, Sweden

Academic thesis for the degree of Doctor of Philosophy, with due permission by the Board of the Faculty of Engineering at Luleå University of Technology, will be defended in public at the university in the Studio, room E632, September 29th, 2004, at 9:00.

External examiner: Dr. Ola Issaksson, Department for Design Methods and Systems, Volvo Aero Corporation, Sweden
To my wife and mother
for their love and support

献给我亲爱的妻子悦
和我尊敬的母亲

Haoxue Ma
Acknowledgements

This work has been carried out at the Division of Computer Aided Design, Department of Applied Physics and Mechanical Engineering, Luleå University of Technology, Sweden. I am grateful for the financial support provided by the Swedish Foundation for Strategic Research through the ENDREA\(^1\) research program and VINNOVA\(^2\) through the Polhem Laboratory at Luleå University of Technology.

I wish to express my deep gratitude to my supervisor, Professor Lennart Karlsson, who initiated this research project and has provided great support throughout the research work. I would like to acknowledge Professor Tore Risch and Dr. Kjell Orsborn at Uppsala Database Laboratory, Uppsala University, for their co-supervision and invaluable support concerning database technology. I also would like to acknowledge Dr. Peter Jeppsson at the Division of Computer Aided Design who supervised me during the first years and led me into this exciting research world. Heartfelt acknowledgement to ALL OF YOU!

I would like to give special thanks to my co-authors and colleagues, Drs. Henrik Johansson and Mattias Nyström at the Division of Computer Aided Design and Mr. Romain Babelon at the Division of Machine Elements, Royal Institute of Technology, for the valuable discussions and successful collaboration we had during this research work. Many thanks to Dr. Peter Törlind for his important hardware support and all my colleagues for building such a pleasurable working environment.

Thanks Wayne Chan for the language proofreading for this thesis and some of the appended papers.

I am also grateful to all my Chinese and Swedish friends in Sweden for the enjoyable spare time we spent together and for becoming my friends for life.

My deep gratitude and love to my Mother and to the memory of my Father for their love, care, education, and encouragement.

Finally, I would like to express deep gratitude to my dear wife Yue Chen. I will never forget the many lonely evenings you endured during the last two years. Your love, patience, support, and sacrifice are undoubtedly a part of this achievement.

Haoxue Ma
September 2004, Luleå

\(^1\) The Swedish Engineering Design Research and Education Agenda

\(^2\) Swedish Agency for Innovation Systems
Abstract

In recent years, the rapid development of computer technologies has greatly impacted the product development process. In the engineering area, powerful computer-based tools such as Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) systems enable engineers to fulfil various design tasks and realise product concepts in the early phase of the product development process, without actually manufacturing physical parts. However, the increasing complexity of modern products and the globalization of product development further necessitate a distributed and collaborative design environment where different computer programs and distributed experts in similar or different domains need to be collaboratively involved on a common design activity. Therefore, the integration and communication of engineering information are, amongst others, two of the most critical technical factors in ensuring successful collaboration.

This thesis investigates the requirements of collaborative design and develops methods to facilitate the integration and communication of engineering information between engineering information systems in such a distributed and heterogeneous collaborative design environment.

A key enabling technology in this thesis is object-relational database technology supporting an object-oriented data model and an object-oriented query language. It is found that modern database technologies not only provide direct support in effectively storing and efficiently accessing CAD data having complex structures, but also assist collaborative design activities by offering advanced database features such as a high level declarative query language, active rules, database wrapper-mediators, and peer database architecture. These features can meet the critical requirements of engineering collaborative design in terms of the integration and communication of engineering information and the interaction between distributed designers. Modern database technology has been extensively used in the thesis and is expected to play an important role in future collaborative design environments. In addition, this thesis employed CORBA – an industry standard for distributed object computing, as a key mechanism to provide object interoperability between CAD systems. Therefore, remote CAD objects and their functionalities are transparently accessed, operated, and distributed through CORBA-based interface across different platforms.

It is found that functionalities of traditional, stand-alone, single-user CAD system can be extended by employing modern database technologies and distributed object computing. These technologies provide cornerstone and effective support in building scalable, extensible, easy-to-maintain, and interactive information systems supporting collaborative design, as illustrated in the demonstration systems developed in the present thesis.

Keywords:
collaborative design, information integration, information communication, computer aided design, database technology, distributed object computing, engineering information
Thesis

This thesis comprises a summary and the following appended papers.


**Paper C.** Ma, H., Johansson, H., and Orsborn, K., Distribution and Synchronisation of Engineering Information Using Active Database Technology, preliminarily accepted by *Advances of Engineering Software*. (Note: it was published at the 7th International Conference on the Application of Artificial Intelligence to Civil and Structural, Egmond aan Zee, The Netherlands, 2-4 September 2003)

**Paper D.** Ma, H., M-Sync: A System for Distributed Collaborative Design, submitted for journal publication.

**Paper E.** Ma, H. and Risch, T., A Database Approach for Information Communication in a Peer-to-Peer Collaborative CAD Environment, submitted for journal publication.
## Contents

Chapter 1. Introduction................................................................................1
  1.1 Engineering Collaborative Design ......................................................3
  1.2 Research Question .............................................................................5
  1.3 Aim and Scope of the Research..........................................................5
  1.4 Research Approach ............................................................................6
  1.5 Thesis Outline....................................................................................6

Chapter 2. Related Work.............................................................................7
  2.1 Groupware in Computer-Supported Cooperative Work ............7
  2.2 State-of-the-Art..................................................................................8
    2.2.1 Product Modelling..............................................................8
    2.2.2 User Interaction ..............................................................10
    2.2.3 System Architecture .........................................................11
  2.3 The STEP Standard..........................................................................13

Chapter 3. Database Technology for Engineering Applications........17
  3.1 Data Models.....................................................................................17
  3.2 Active Rules.....................................................................................19
  3.3 Peer Database ...................................................................................20
  3.4 Data Integration ...............................................................................21
    3.4.1 Data Warehousing..............................................................21
    3.4.2 Wrapper-Mediator Approach............................................21
  3.5 AMOS II..........................................................................................23

Chapter 4. Distributed Object Computing........................................27

Chapter 5. Summary of the Appended Papers................................31
  5.1 Paper A............................................................................................32
    5.1.1 Summary...................................................................................32
    5.1.2 Contribution.............................................................................33
  5.2 Paper B ............................................................................................33
    5.2.1 Summary...................................................................................33
    5.2.2 Contribution.............................................................................34
  5.3 Paper C............................................................................................34
    5.3.1 Summary...................................................................................34
    5.3.2 Contribution.............................................................................36
  5.4 Paper D............................................................................................36
    5.4.1 Summary...................................................................................36
    5.4.2 Contribution.............................................................................37
  5.5 Paper E ............................................................................................37
    5.5.1 Summary...................................................................................37
    5.5.2 Contribution.............................................................................38

Chapter 6. Conclusions and Future Work ........................................39
  6.1 Future work......................................................................................40
Chapter 1. Introduction

Keen competition in modern industry motivates companies to produce better products with shorter lead-times at less cost. In recent years, the rapid development and implementation of computer technologies have greatly impacted the product development process. Advanced computer-based tools such as Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) systems are widely-used in the engineering field. They enable engineers to design, analyze, simulate, and test products by means of digital prototyping in the early phase of the product development process without actually manufacturing physical parts. Therefore, these tools dramatically help companies to produce better products with shorter time at less cost.

However, the increasing complexity of the modern product and the globalization of product development further necessitate a distributed and collaborative design environment. Here, several computer programs supporting various engineering activities and co-located designers originating in similar or different domains need to be involved and interacting in collaborative work on a common design activity. For instance, during automobile or construction equipment development, most design activities use CAD solid models as common bases. These CAD models are then used in other downstream activities as illustrated in Figure 1: the structural analysis is performed with finite element methods (FEM) while the suspension is analysed by multibody system (MBS) analysis. Body development also needs computational analysis to predict the airflow by using computational fluid dynamics (CFD) tools. Therefore, the integration and communication of engineering information are, amongst other things, two of the most critical technical factors in ensuring successful collaboration.
Figure 1. Collaborative work involving several engineering domains

It is technically difficult and time consuming to integrate and communicate information between computers. Two of the most important reasons are (1) the diversity of computer software and hardware and (2) the distribution of both computers and users:

- Heterogeneity is the inevitable result caused by the diversity of computer software and hardware. Nowadays, computers are widely-used in various trades and professions ranging from traditional business management to advanced scientific and engineering computing. Software packages running on diverse computers have been developed for different purposes and used to fulfill different disciplinary tasks such as office automation, computer aided design, and computer aided manufacturing. These programs use their proprietary formats - domain specific or application specific - to represent and store information and run on different operating systems and hardware.

- Computer network technology has led to a distributed working environment where an ever increasing number of distributed information systems are involved and running on many computers. These computers are connected over networks having different architectures, protocol standards, bandwidths, etc. Some typical domains using this type of application are electronic commerce, airline reservation systems, banking systems, and manufacturing control systems.
1.1 Engineering Collaborative Design

To motivate and guide this research work, we must first identify the limitations of current information systems and the requirements of engineering collaborative design. Engineering collaborative design is a very broad term that is closely related to several technical and social disciplines such as engineering design, computer science, human computer interaction, decision-making, behaviour, and social sciences. In this thesis, the term “engineering collaborative design” refers to a design activity where distributed participants in a design team jointly perform engineering design tasks towards a common goal with the help of computer based tools and computer network such as CAD systems and the Internet.

It is actually even more difficult to integrate and communicate engineering information between distributed engineering information systems due to some characteristics, besides information heterogeneity and distribution, that are peculiar to the engineering information and engineering collaborative design environment. Some of these characteristics are, for example,

- Efficiently managing and manipulating huge amounts of information having complex representation, complex operation, and long-duration transactions.

- Distributed participants need to collaborate during the design process. Some necessary collaborative activities are, e.g. real time interaction between participants, timely exchange of relevant information between different software and hardware, capturing changes made by other participants, and access control to shared information.

- Maintaining consistency between distributed data sources is also a very important issue. Failure on data consistency control may lead to invalid or even result in a completely wrong design.

The evolution of CAD systems has experienced from drawing CADs, wireframe CADs, surface CADs, and solid CADs, i.e. from 2D drafting to 3D solid modelling [1]. Since the beginning of the 1980s, CAD technology has become more economically attractive and evolved into a fully developed tool that is now widely used in industry. Modern CAD systems are much more than powerful tools drawing lines by electronic means. They allow engineers to realize and maintain the design intent from concept through manufacturing by means of design, analysis, test, and product simulation in a virtual environment. Some popular commercial CAD systems on the market include Unigraphics and I-DEAS from UGS [2], SolidWorks and CATIA from Dassault Systèmes [3], and Pro/Engineer from Parametric Technology Corporation [4].

However, the original design philosophy of a CAD system is to support individual work in resolving increased product complexity. Despite much progress, current CAD systems still have inherent limitations concerning collaborative design. Some typical limitations are,

- CAD systems are stand-alone, single-user applications.
Single-user access to data and associated information, such as mechanisms for a CAD model check-in and check-out, means that users have insufficient interaction and collaboration in terms of online CAD model co-design.

Collaborative design participants have to coordinate their design activities by other means, e.g. telephone, email, or video conferencing, rather than in their current CAD environment.

- CAD system uses file-based storage and data exchange.
  - This produces some drawbacks in terms of access to CAD data structure and high-level functionality.
  - Parallel access to shared information is restricted.
  - If product data is distributed amongst several files, accessing and combining the required part of the file is difficult.

Many companies use Product Data Management (PDM) systems to manage their product data. PDM systems provide mechanisms to manage documentary product data and relationships between them so that it is easier to access, refer to and reference. They can also manage workflow and work history by keeping track of required data and supporting documents and by using the mechanisms that provide revision, approval and release management. However, current PDM systems have insufficient capabilities to solve many problems raised in engineering collaborative design in terms of integration and communication of engineering data such as CAD model data.

Considering the above observations, it is concluded that successful information systems supporting engineering collaborative design should fulfil the following high level requirements:

- Provide participants a shared or autonomous workspace fully capable of performing design tasks on all levels.
- Have effective storage that can naturally represent and efficiently manipulate CAD model information having complex representation, complex operation, and long-duration transactions.
- Provides functionalities enabling geographically distributed designers to timely communicate and interact with each other in synchronous or asynchronous manners. In other words, designers should be able to explore and experiment with alternative ideas or share relevant product data during the design phase. This requirement can be further decomposed into several sub-requirements:
  - Support for a decentralized system architecture, since the trend of engineering information systems has evolved from centralized to decentralized systems due to the advent of computer network technology and enterprise globalisation.
Integration and Communication of Engineering Information in Collaborative Design

- Share product information between different design and analysis tools in a system and hardware independent manner.
- Coordinate design activities and results by providing effective transaction management and consistency control between distributed systems.
- Enable some degrees of design version/history management.
- Be aware of the other participants’ design stages via visualisation tools such as 3D CAD model viewer.

1.2 Research Question

The observed limitations of current information systems and the requirements of engineering collaborative design have led to the following formulated research question of this thesis:

*Which tools, methods, and architectures are needed to improve the information systems supporting engineering collaborative design?*

This is a broad research question that can be further decomposed into a few major sub-questions, namely

- How should huge amounts of complex engineering information that are distributed, heterogeneous, dynamic, and continuously evolving be efficiently managed and manipulated?
- How is the information integration and communication between engineering information systems improved?
- How is work collaboration between distributed participants assisted?

1.3 Aim and Scope of the Research

The overall aim of this thesis is to improve the ability and efficiency of information management in an engineering collaborative design environment, and thereby shorten the product development lead-time. More specifically, the aim is to develop tools and methods

- To efficiently integrate and communicate engineering information between distributed and heterogeneous engineering design and analysis tools
- To facilitate interactive collaborative work between distributed participants.

This thesis is grounded in the field of mechanical engineering and draws attention to some technical issues of information management for engineering collaborative design. It is not the intention of this work to develop new software, industry standards, or any data communication protocols. Instead,
existing IT tools such as modern database technologies and distributed object computing are used to achieve the research aim. It is expected that the findings can improve the information systems concerning those features for engineering collaborative design.

**1.4 Research Approach**

This research work employs database management systems, distributed object computing, and commercial CAD systems to build an integrated engineering information environment supporting collaborative design.

The database technology used facilitates the information management and interactive work collaboration between users by offering unique database features such as an extensible object-oriented data model, a powerful declarative query language, active database rules, peer-to-peer database architecture, and a wrapper-mediator approach. Creation, distribution, and management of distributed objects in a network are supported by distributed object computing. CORBA [5], a well-known specification for distributed object computing, is used to realize information integration in a platform, hardware, and software-independent manner. Furthermore, existing commercial CAD systems have been used in this work. The advantage of using commercial CAD systems is that their powerful capability enables users to perform complex design tasks. They normally also have a mature and well-defined interface enabling users to access and manipulate its internal data. Moreover, they are widely used in industry, and development based on such systems can therefore be easily recognised by industry.

The proposed methods were implemented and validated by constructing different prototype systems presented in the five research papers. Each paper focuses on specific technical problems and contributes pieces to the overall goal of the work.

**1.5 Thesis Outline**

In Chapter 2, related work including groupware and the standard for the exchange of product model data is investigated and classified according to product modelling, user interaction, and system architecture.

Enabling technologies adopted in this work are presented in Chapters 3 and 4. The motivation of adopting these technologies is illustrated through the introduction and discussion of their unique features and requirements of collaborative design that they can meet.

The summaries, results, and contributions of co-authors of each appended paper are presented in Chapter 5. The relationship between these papers regarding technologies adopted and causality is also outlined.

Finally, the conclusions of this thesis are summarised and future work is identified in Chapter 6.
Chapter 2. Related Work

2.1 Groupware in Computer-Supported Cooperative Work

Computer-Supported Cooperative Work (CSCW) is a research field that uses Internet and distributed system technology to support the work of a group of people who are often physically distanced from each other [6, 7]. CSCW began with the advent of networks and multi-user operation systems and can be traced to the NLS system developed by Engelbart and Lehman [8].

Groupware is computer software designed to facilitate collaborative activities in CSCW [9, 10]. It can be defined as “computer-based systems that support groups of people engaged in a common task (or goal) and that provide and interface to a shared environment” [11]. As a key enabling technology of CSCW, groupware must to some degree support features such as communication between participants, collaboration in a shared information space, and coordination of collective contributions [12]. Typical groupware applications include electronic mail (e-mail), video conferencing, shared document manager, shared whiteboard, and application sharing.

To some extent, groupware meets collaborative work requirements since it provides functionalities allowing users to exchange and share product information. This can be achieved by either sending product model files via email or performing real-time multi-media (video, picture or sound) communication via video conferencing, shared whiteboard, and application sharing. Therefore, “what you see is what I see” can be achieved between co-located designers. Design history, i.e. mailing history in an e-mail system, can be automatically maintained. Some even offer capabilities to remotely update product information one user at a time.

Using groupware to communicate typically occurs as follows: when local information is requested by remote designers, a local designer must first save the design work and then send a copy of the current work to the remote designers by either e-mail or FTP, or by using a video conferencing program, shared application, or whiteboard to share screen images of the current work with other designers. Since only rudimentary mechanisms are provided to manage shared design objects in collaborative design, interactivity of this working manner is limited. It also lacks indexing functionality – imaging of how one
would feel to search a tape for 2 hours for only 3 minutes of interesting content. More importantly, CAD models contain large amounts of high-level design information. Performing real-time online discussion and modification on such information are essential for designers. This is especially apparent when designers perform closely-coupled collaborative design tasks. However, information sharing supported by groupware is on either a static or “view” level, since groupware has no or at least insufficient functionalities to support collaborative product design in a concurrent and interactive manner.

2.2 State-of-the-Art

Besides groupware technology in a CSCW community, information systems supporting distributed engineering collaborative design have been studied and developed in recent years. This sub-section gives a brief overview of them according to product modelling, user interaction, and system architecture.

2.2.1 Product Modelling

With the continuous development of programming languages, efforts in product modelling have been made to improve the share of product data in collaborative design environments. Many systems developed in recent years employ VRML (Virtual Reality Modelling Language), XML (Extensible Markup Language), Java applet, and Java 3D to represent product data. Feature-based modelling is also another popular issue in the collaborative CAD/CAM research community.

VRML is an abbreviation for Virtual Reality Modelling Language, or previously called Virtual Reality Markup Language [13, 14]. Unlike a conventional programming language like C/C++, VRML is a WWW compatible and platform independent language describing the geometry and behaviour of a 3D object for onscreen virtual reality environments. VRML can produce a 3D space on screen from a certain viewpoint and then, by mouse movement and clicks, create the illusion that you are moving through a real space by gradually changing the viewpoint position. The main advantages of using VRML is its ability to represent 3D animations and both static and dynamic 3D data, as well as it being an interactive publicly available standardized format that is supported by a number of CAD modelling tools [15].

Many systems use VRML to share 3D objects in collaborative work. TeCo3D [16] enables collaborative designers to share interactive and dynamic 3D models defined in VRML. 3D-Syn [17] is a 3D collaborative mechanical part modelling system. Employing Java applet-server architecture, users can use it to view and manipulate a 3D part model presented by using VRML with a collaboration applet at the client site. A user’s manipulation of part objects is transparently and instantaneously transmitted to the others. CyberEye [18] uses a Java 3D-based 3D modelling browser, CyberEye Viewer, to share VRML files stored on a server. Access to VRML files is implemented using Java applets.
Integration and Communication of Engineering Information in Collaborative Design

Other systems [19-21] also employ VRML as an intermediate to represent and share 3D objects in multi-user design environment.

XML is a pared-down version of SGML - Standard Generalized Markup Language (ISO 8879) [22] - designed especially for Web documents [23, 24]. It is a meta language containing a set of rules to construct other markup languages for different types of documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications. An XML document can be a file or a data stream containing nested elements starting with a root element. Its metadata description defines structures and constraints for a particular type of XML documents using Document Type Definitions. Therefore, XML provides ways to describe and store complex data structures suitable for exchange over the Internet.

Su et al. [25] proposed an open collaborative CAD system model based on Java/CORBA/XML. XML is used to describe features of and relations between solid model primitives, as well as design a language to define feature design history. In this system, sending XML-based semantic messages supports distributed design activity, thereby decreasing the necessary data volume transferred via the network. Peng et al. [26] presented a prototype implementation of an engineering data access system for a finite element analysis program - OpenSees. In this system, XML is chosen as the external data representation for exchanging data between applications. Matrix service was developed to convert matrix-type data into an XML element while XML packaging service packages both XML elements and basic-type data into XML files. Mitschang [27] developed a system called TOGA and a data propagator called CHAMPAGNE that jointly realizes a shared information space. In CHAMPAGNE, adaptors is an important model used to achieve information sharing between different systems, by mapping both changed data objects of external information systems into an XML representation and XML-represented output data back to the external systems.

Feature-based modelling is another popular topic for product data modelling in collaborative design [28-31]. In feature-based modelling, product models contain not only geometric information, but also features such as the function of product part for the end-user or the way product part can be manufactured or assembled. Feature-based modelling is based on advanced geometric modelling such as parametric and constraint-based modelling. In the NetFEATURE system developed by Lee et al. [32], feature-based modelling is combined with network technologies to support product modelling in collaborative design. A neutral feature model server provides the functionality needed for feature modelling and supports multiple views for connected clients. Feature-based design, feature recognition, and process planning can be run at the client side, where each client can view and extract the necessary data required in its application context from the neutral feature model. Bidarra et al. [33] presented a web-based collaborative feature modelling system called webSpiff. It has a client-server architecture where the server provides feature validation, multiple
views, and visualization facilities through session manager and Spiff modelling system, while clients provide user interface to display feature model images and select feature faces.

2.2.2 User Interaction

Collaborative user interactions are typically classified based on a taxonomy that distinguishes four categories: same time/same place, different time/same place, same time/different place, and different time/different place [11]. The two latter categories are known as synchronous and asynchronous distributed interaction. In synchronous interaction, geographically dispersed users collaborate in real-time over a network. Shared information in the collaborative design can be jointly and simultaneously retrieved, filtered, accessed, and modified by any authorized users. Changes introduced by one user are immediately broadcasted to the others. It is therefore an important issue to study mechanisms for controlling user co-modification on shared information and consistency of distributed data.

Qiang et al. [34] proposed a system, called WPDSS, to support CAD-based synchronous collaborative design through the Web. In this system, CAD macro files were used to exchange information among team members. Two modules, viz. the site co-modifying interface model and the server co-modifying interface model, were proposed to realize the monitoring methodology for synchronising group operations, and maintaining consistency of shared CAD models. The monitor issues modifying authority to one user in each round. Ramani et al. [35] proposed a thin-client collaborative system, called CADDAC, for shape conceptualization. It allows the creation and modification on geometry in a collaborative and synchronous mode, and maintains the shape creation history in the database on the server-side. Only the client who has the control, called master-client, is capable of doing real-time editing on geometry over the network. Nam and Wright [29] developed a system called SYnchronous COllaborative 3D CAD (Syco3D), which allows a small team of distributed designers to work together to build and edit virtual 3D models. A real-time collaboration module, called Shared Stage, was proposed and used as a means of transferring and referencing 3D models across different users’ workspaces. Together with synchronised representation and multi-user pointers, the system enables designers to collaborate and interact synchronously. Other technical aspects, including algorithm development, of synchronous system have been addressed in [36-39].

However, collaborative work does not always require real-time communication or simultaneous interaction. Designers who sometimes structure their work without being restricted by a schedule can contribute independently to a shared product. The different versions of design are assembled at the end to identify the optimal design process and design object. A well organized, shared information repository is thus needed where designers can place their contributions and retrieval tools to find information created by others [12].

Integration and Communication of Engineering Information in Collaborative Design
Xie and Salvendy [40] designed a prototype 3D CAD browser to support the use of asynchronous collaboration. Four agent-based features, viz. Comment Status, Communication Agent with Instant Comment Alert, Team Member Information Enhancement, and Task Information Enhancement, were proposed to support communication, activity coordination, and collaboration awareness issues, e.g. reading and processing every incoming comment, informing others about the comments, and knowing the status of the collaboration tasks and their team members. Edwards et al. [41] presented an infrastructure, called Bayou, designed to support the construction of asynchronous collaborative applications. This system addressed a number of features concerning asynchronous collaboration, such as application-defined conflict detection and resolution, selection of session guarantees, selection of committed or tentative data, and replica selection. These features are presently implemented in applications such as mail user agent, group calendar, and project coordinator.

Some systems [17, 42-45] address both synchronous and asynchronous interaction – called hybrid interaction. Among them, Kim et al. [17] presented a 3D-Syn system where both 3D synchronous communication and asynchronous coordination were supported. A user’s action on a 3D part model is transparently sent and displayed to other users in real-time. The asynchronous coordination is simply supported by a design history log file that is automatically recorded by the system and launched to the client’s browser under request. Roller et al. [45] developed a version model and a transaction group model as an integrated transaction concept to support synchronous and asynchronous cooperation for collaborative design activities. The version model supports asynchronous workspaces among designers who work on different object copies (multi-copy approach), while the transaction group model supports synchronous cooperative work (single-copy approach).

2.2.3 System Architecture

There are basically two kinds of system architectures for distributed collaborative systems, i.e. the centralised model, referred to as client/server architecture where all shared data is maintained at a single location, and the replicated model, referred to as peer-to-peer architecture where each site maintains and processes a copy of shared data, as shown in Figure 2. Most current collaborative design systems employ client/server architecture.
In the centralised model [34, 35, 38, 46], a central database, i.e. server, is employed to manage engineering information. Co-located designers download design information such as a CAD model from the server and edit the model using local design programs. Figure 3 shows the typical work procedure for this type of collaborative work.

If a certain designer wishes to edit a CAD model, they must first copy the model into the local system (check-out), and then start the editing task. During editing, the model is locked by the central database system and can only be modified by the current designer. If the designer has modified the model, they must update the corresponding model stored in the central database (check-in), unlock the model, and send a message to the other designers regarding this update. Other designers must therefore re-download the modified model to proceed to the design task. The key advantage of the centralised model is that it is easier to ensure data consistency as there is only one master copy saved in the central database. However, a problem is that designers only have either shared view or individual access to CAD models, meaning that one designer has to wait until other members release the editing right to change the model in the
Performing collaborative work using this model typically requires high network bandwidth, since in most systems the whole model has to be re-downloaded by other designers even if there is just a minor update made on the master model, e.g. one dimension of a solid model. It is also potentially less fault-tolerant than a replicated model because the central host is a single point of possible system-wide failure [10].

The replicated model [47-50] does not have a central database containing master models. Instead, each node contains a copy of the shared data in the local system; hence P2P eliminates the single-source bottleneck and designers have full access to the data at any time. Each node also has equivalent capabilities as well as responsibilities that either request or provide service, differing from the centralized model where the central database is dedicated to serving the others. Modifications made to the local model are identified and then propagated to other collaborative designers in the form of native format, standardized format, or semantic design messages. A reduced data volume is distributed since only the modifications are transferred rather than the whole modified model. The various sites could be synchronised by applying, interpreting, or executing the data or design message sent by the modifier. Additionally, designers can perform independent simultaneous work and have different views of the shared data since the data reside locally; therefore, designers can later modify the model and merge the changes with others. However, compared with the centralized model, maintaining data consistency among distributed replicas is more complex while the number of propagating messages is much larger.

### 2.3 The STEP Standard

As an important related technology, an industry standard named the Standard for the Exchange of Product Model Data (STEP) [51] is mentioned in this section.

Since engineering applications generally have their own proprietary file formats, data generated by one application normally cannot be directly read by other applications. One way to exchange data is by direct translation between the applications, as illustrated in Figure 4 (a). Compare to neutral file exchange described later, this method often yields good exchange results. The translators can be optimised to perform a single task and include a more complete set of information required by data exchange. However, there are \(n \times (n-1)\) possible translators between \(n\) systems since a unique pair of translators is needed for every version of every combination of applications. It is expensive to maintain all these translators.
To solve these problems, considerable amounts of effort have been expended in developing standards for data exchange. STEP, an ISO standard aimed at facilitating the exchanging and sharing of the total information used to define an industrial product, is one well-known outcome from the effort. It is intended to neutrally and system independently describe products throughout their lifecycle including engineering, manufacturing, and support data [52-54]. The nature of the STEP standard makes it suitable not only for neutral file-based data exchange, as shown in Figure 4(b), but also as a basis for implementing and sharing product databases and persistent archiving, as shown in Figure 4(c).

STEP consists of a group of parts that covers topics such as methods used to represent the standard as well as data structures for specific domains. For instance, part 11 defines EXPRESS [55] - the information modelling language used to specify the normative part of all the information models in STEP, while AP203 [56] defines specifications for solid geometry.

EXPRESS is a conceptual schema language based upon the entity-attribute-relationship model with generalization and constraint-specification constructs. It is an object-oriented model specification language used to define a representation of product data [57]. It is not a programming language, but rather an information modelling language to define entity data structures, relationships, and constraints on data. There are tools that use the entities and types defined in the EXPRESS definition, the so called EXPRESS schema, to build corresponding classes in object-oriented programming languages such as C++ and Java. The Standard Data Access Interface (SDAI) [58] is an interface that provides a standard set of data access functions to manipulate EXPRESS defined data. STEP uses SDAI to address the storage and retrieval of data between an application and a database according to an EXPRESS schema. The functional specification of the SDAI is independent of any implementation language, since it is realized by using SDAI language binding, such as bindings for C, C++, and Java.

STEP is a well-known international standard for the exchange of product data. Most CAD/CAM systems have mechanisms to import and export a STEP file to exchange data. However, STEP implementation tools were initially
designed for data exchange via static files and lack support for concurrent design activities [59]. More specifically,

- EXPRESS is a data description language describing an information domain in terms of entities, attributes, and relationships, i.e. there is no functionality for accessing or manipulating the data.

- SDAI allows access to STEP repository but it does not offer the appropriate functions to provide high-level access and to manipulate CAx models. This means the functionality offered by SDAI is not sufficient for online CAx interoperability.

- File-based data storage and exchange have some drawbacks:
  - No direct support for a query language to access the data in files.
  - Parallel or random access to shared information is restricted.
  - Lacks coordination between files. If the product data is distributed between several files, accessing and combining the required part of the file is difficult.
  - Data integrity in the case of concurrent access cannot be guaranteed.

- SDAI has (if any) limited support for accessing and sharing distributed and fragmented information.

- Loss of information as with any other standards.

Much research [60-66] has recently started to address some of these problems, with most of them employing Database Management System (DBMS) to overcome the above STEP limitations. In the next chapter, DBMS features relevant to this research work will be introduced.
Chapter 3. Database Technology for Engineering Applications

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. Elmasri and Navathe [67] defines a DBMS as

* A general-purpose software system that facilitates the processes of defining, constructing, and manipulating databases for various applications

DBMS’s are extensively used in almost all areas where computers are used, providing a convenient and efficient environment for users to store and retrieve information. Some typical functionalities of DBMS are, for example,

- persistent storage for a large amount of data
- population and manipulation on data stored in the database
- transaction management ensures that the database remains a consistent stage
- a recovery technique to restore the database in the case of failure

In the next sections of this chapter, some advanced DBMS features employed in this work are presented.

3.1 Data Models

The most common data model, i.e. relational data model, uses a collection of tables called relations to represent both data and the interrelationship between the data. Table is a two dimension uniform data structure as shown in Table 1. Each table has columns with unique names called attributes and rows called tuples. There must be at least one attribute called key that is unique to each tuple in a table. In this example, the relation *employee* has four attributes, viz. *ID*, *Name*, *Age*, and *Salary*, with the key attribute being *ID*. A relational database management system (RDBMS) uses the relational data model to define schema and store data in the database, and has been widely used to handle administrative and business information. Even today, RDBMSs are still the most popular and widespread on the database market. Some representative applications are banking, airlines, finance, and human resources.
In an RDBMS, data manipulation is usually made through a high-level query language - Structured Query Language (SQL). The objective of SQL is to provide both programming specialists and inexperienced users with a declarative high-level language for relational database rather than using procedural programming to manipulate data. In 1986, ANSI and ISO published an SQL standard, called SQL-86. Today’s most recent version is SQL-99 [68]. SQL has clearly established itself as the standard relational database language [69] and is actually one of the most important contributing factors to the great success of RDMBS. Although SQL is referred to as a query language, it can do much more than just query a database. For example, it can be used to define data structure and update data in the database.

In recent years, demand has grown for ways to deal with more complex data types, such as in engineering and scientific applications. These data types usually require efficient management and manipulation of information with, e.g. complex representation, operation, need for high extensibility, and long-duration transactions. A schema of a CAD view is illustrated in Figure 5. However, the relational model is essentially a record-based model that is often limited or can rarely handle such complex data types. Object-oriented database management systems (OODBMS) were proposed to meet the certain needs of these applications. OODBMS support a richer and more extensible data model, called object-oriented data model. Basic object-oriented concepts usually supported in OODBMS include objects, object identity, composite object, methods, encapsulation, type hierarchies and inheritance, operator overloading, late binding, and version and configuration management [70].

OODBMS use standard query language - Object Query Language (OQL) - to query data. OQL is designed to tightly integrate with its corresponding object-oriented programming languages where an OQL query can be embedded. OODBMS provides persistent storage of programming objects, i.e. the objects are persistently stored in secondary storage and can later be retrieved even if the application terminates. However, it is limited and rigid in terms of built-in hierarchical paths in the data [67] and does not include several RDBMS features such as a high-level declarative query language, meta data management, and views.

The current research trend in the database community focuses on object-relational database management systems (ORDBMS) [71]. ORDBMS combine many features of the object data model and languages into the relational data model. Therefore, it contains features from both RDBMS and OODBMS, such
as object identifier, query language supporting object-oriented data models, and triggers. SQL-99, a.k.a. SQL3, is an ISO standard language based on an earlier version of SQL. It not only includes an object-orientation methodology, but also other features like triggers, procedural language, and error handling. Companies in the RDBMS market, e.g. IBM and Oracle, have released products in this area. Many research prototypes have also been built, with some of the most well known being HP-Iris, Berkeley-Postgres, and IBM Starburst.

![Diagram of CAD view](image)

**Figure 5. Schema of CAD view**

### 3.2 Active Rules

Traditional database management systems are passive, i.e. data is created, retrieved, modified, and deleted only in response to operations issued by users or application programs. Active database management systems (ADBMS) are event-driven DBMSs that enhance the functionality of traditional DBMSs so that certain operations can be automatically performed in response to the occurrence of certain events or when certain conditions are being satisfied [72]. Active database technology uses active rules, so called event-condition-action (ECA) rules, as an integral feature to monitor events specified in the rules declaration. When the occurring event satisfies the conditions, the corresponding actions are executed. Hence, active database systems can recognise and react to specific situations without direct and explicit user or application requests [73]. Furthermore, the rules can be shared by many applications while the database systems can optimise their implementation [74].

ECA rules are often declared as conditional expressions, like *if-then-else*, or case statements in conventional programming languages. However, such statements are static, i.e. they cannot be changed unless the code is recompiled.
The ECA rules can be dynamically added and changed in databases supporting incremental recompilation of rules and functions. Furthermore, an important difference is that the if-then-else or case statements in conventional programming languages are just parts of sequential code, while ECA statements are executed as event-driven programming, i.e. when an event happens and a condition is true, some code is executed.

The following simple example illustrates ECA rule usage. This rule updates the material of a given part \( p \): when an update of function \( \text{state}(m) \) occurs and condition is true, procedure \( \text{updatePartMaterial} \) will be called to update the material of the part.

```
create rule updateMaterial( part p ) as
  from material m
  on updated( state( m ) )
  when m = madeOf( p ) and updateState( m ) = true
  do updatePartMaterial( p );
activate rule no_high(:cylinder);
```

The use of ADBMS to support different engineering applications and monitor and control engineering models in databases has been discussed and investigated. Sköld [75] specified the requirements demanded by computer integrated manufacturing and telecommunications networks on database technology and presented some scenarios using ADBMS within these two fields. Andler [76] introduced a method for distributing functions in a real-time control system using ADBMS. Roller et al. [77] developed an active database approach to increase design process productivity by informing designers about the correlations to parallel work and by practising collaborative work. Other works [78-81] further demonstrate the usage of ADBMS within different fields such as process industry, software integration, and workflow management.

### 3.3 Peer Database

Initially used for distributed file sharing and made popular by the Napster system [82], Peer-to-Peer (P2P) is a general decentralised software architecture that acts as an alternative to the traditional client-server architecture. P2P can be simply defined as the sharing of computer resources and services by direct exchange [83], meaning that a P2P network directly distributes information among the peers instead of concentrating it at a single server, as in a client/server architecture. It opens a new window for, amongst others, information communication over the network due to its fundamental principles, namely resource sharing, decentralization, and self-organization [84, 85]. Typical P2P applications are found in file sharing systems, distributed computations, and instant messaging systems.

According to Özsu and Valduriez [86], how DBMSs have been distributed can be classified into two classes, i.e. client/server architecture and P2P architectures. In a client/server architecture, the functionalities of client and
server are different, i.e. the servers concentrate on data management for clients and the clients provide an application environment including user interface. In P2P systems, all nodes are equivalent, i.e. they all have full DBMS functionality and can equally communicate amongst each other by both providing and requesting services. A peer database network normally consists of different types of autonomously managed databases at peers and the use of independent local schemas.

3.4 Data Integration

In general, data reside in different sources. A typical data source could be a file, a web page, a CAD system, or an RDBMS. The aim of data integration is to hide the technical details and complexity of each individual data source for the users and provide users a single interface to the data in the data sources. Basically there are two main ways to integrate heterogeneous data, i.e. data warehousing where all data is ‘pumped’ into a central database and the wrapper-mediator approach where the data is kept in the sources.

3.4.1 Data Warehousing

Data warehousing is a centralized way to integrate heterogeneous data from different data sources. It is conceptually close to the shared STEP model shown in Figure 4 (b). In this approach, data is periodically extracted from different data sources, transformed into a uniform representation and then stored into a huge central repository, called data warehouse. Data warehouses therefore contain a wide variety of data that present a coherent picture of business conditions at a single point in time and are normally used by large organisations. However, the data warehouse approach is unpractical for online engineering collaborative design because data is not always up-to-date. All data is copied into the data warehouse and is refreshed from time to time so that some degree of delay between the current data in the sources and data copy in the data warehouse always exists.

3.4.2 Wrapper-Mediator Approach

Another way to integrate heterogeneous information is to use the database wrapper-mediator approach introduced by Wiederhold [87]. The goal is to integrate heterogeneous data by adding an intermediate software layer, called a wrapper-mediator layer, between both data sources and applications to provide users an integrated global view of the data sources as if they are a single entity. Its architecture is shown in Figure 6.
Wrappers are specialized software components that provide access to particular kinds of external data sources. The typical functions of wrappers are to retrieve the data of sources using a common query language and translate it into mediators’ common data model (CDM). Hence, wrappers provide a uniform representation of all data sources in terms of the CDM. Mediators provide coherent views of the data in the data sources by performing semantic reconciliation of the CDM data representations provided by wrappers [88] and share them with higher layers of mediators or applications.

When a user issues a query, a mediator decomposes the query into smaller data source-specific queries and lets the wrappers execute them. After the decomposed queries have been executed, the wrappers then sent the results back to the mediator, which finally integrates them into the final result of the query issued by the user.

The wrapper-mediator layer appears as a single virtual database unit to all integrated sources. Its architecture differs fundamentally from a data warehouse in that integrated data does not need to be materialized in the local database, i.e. all data are kept at the sources and all data access and transformation are performed on the fly during query execution. This approach presents a number of easily identifiable advantages:

- The specialized semantic integration can be applied to any kind of data (via wrapper) and can query most current data
- Such a distributed system is extensible and flexible due to its specialized components
- It has high-level currency since data is kept in the sources and accessed online
- Respects resource autonomy
• Light-weight since the mediator system itself need not contain much data, while a data warehouse becomes a very large database.

According to the requirements of engineering collaborative design as mentioned in section 1.1, it is thought that the database wrapper-mediator approach is preferred to data warehousing when integrating heterogeneous data.

### 3.5 AMOS II

AMOS II (The Active Mediator Object System) is an active, object-relational, peer database management system. The purpose of the AMOS II project is to develop and demonstrate a mediator architecture to support information systems where applications and users combine and analyze data from different and distributed data sources, such as a conventional database, text files, data exchange files, or WWW pages. Its system architecture is illustrated in Figure 7.

![AMOS II system architecture](image)

**Figure 7. AMOS II system architecture**

As a central enabling technology, AMOS II is used in all five research papers of this thesis. This section gives an overview of some important system features.

- The AMOS II kernel is an open and extensible DBMS supporting object-oriented data model. The AMOS II data model consists of three basic concepts - objects, types, and functions - corresponding to instances, classes, and methods in an object-oriented programming language.

  - Objects model all entities in the database, i.e. everything in the database is represented as an object managed by the system. There are two main kinds of representations of objects, viz. literals and surrogates. The literals are self-described system-maintained objects that do not have explicit OIDs, while surrogates have associated OIDs explicitly created and deleted by the user or the system.

  - Objects are classified into types, making each object an instance of one or several types. The types are organized in a multiple inheritance,
supertype/subtype hierarchy. If an object is an instance of a type, then it is also an instance of all the supertypes of that type.

- **Functions** are used to model properties of, relationships between, and operations on objects. Basic functions can be classified into stored, derived, foreign, proxy functions, and database procedures. Stored functions represent properties of objects stored in an AMOS II database and proxy functions represent functions in other databases. The derived functions are functions defined in terms of functional queries over other AMOS II functions, while database procedures are defined by using a procedural sublanguage of AMOSQL. Foreign functions provide the low level interfaces to wrap external systems and are implemented through an external language such as C, Java, or LISP.

- AMOSQL is an object-oriented query language for AMOS II used for defining, populating, querying, and updating the database. AMOSQL is compatible with the object parts of SQL-99 and is based on the functional query languages OSQL [89] and DAPLEX [90], with extensions of mediation primitives, multi-directional foreign functions, late binding, active rules, etc. [88]. Compared with using conventional programming language such as C++ and Java, employing such built-in features enables a more efficient development of a more flexible, less complicated, and easier to maintain application.

- AMOS II is an active DBMS [91]. The rules are first-class objects and of the type “rule”. AMOS II supports ECA, event-action (EA), and condition-action (CA) rule models. The AMOS II rule processor handles rule creation, deletion, activation, deactivation, tracing, and execution with the corresponding AMOSQL commands: create rule, delete rule, activate rule, deactivate rule, trace_rules, and ruleCheck. The events that can be specified in AMOS II are updated, added, removed (i.e. monitors updates to stored and derived functions), and created and deleted (i.e. monitors the creation and deletion of an object instance). Events can also be composite. The active rules monitor the “net effect”, i.e. value changes rather than actions, considered an important feature.

- AMOS II is a peer mediator database system, since every mediator peer can act both as a client and a server to any number of other mediators [92]. The mediator servers communicate over the Internet using a TCP/IP socket-based protocol. Each mediator is an autonomous object-relational DBMS with its own DBMS facilities, such as a local database, a data dictionary, a query processor, transaction processing, and remote access to databases. Already implemented AMOS II wrappers include those for relational data sources [93], XML [94], and STEP/EXPRESS [60]. Its mediation architecture is illustrated in Figure 8.

- AMOS II is a lightweight main-memory resident DBMS, i.e. the entire database resides in primary memory. The performance of such a DBMS is
considerably higher than a secondary, storage-based conventional DBMS, making it suitable to embed AMOS II into or interface it with other systems such as a CAD system. The functionality of a system can therefore be extended with DBMS facilities such as storage management, uniform data model, query language, and query processing.

The features of AMOS II make it possible to represent, store, access, and manipulate complex engineering data from different and distributed data sources, along with timely reactions to specific events. The usage of these features in the appended papers is summarised in Chapter 5.

Figure 8. AMOS II mediation architecture
Chapter 4. Distributed Object Computing

The Common Object Request Broker Architecture (CORBA), which is being standardized by the Object Management Group [95], is one of the most important distributed computing infrastructures to support the integration, sharing, and exchange of information, especially concerning object technology. Its main objective is application interoperability and portability in a heterogeneous network environment, i.e. allowing the objects described in any language to be shared across different OS and platforms. It is widely used in the areas of manufacturing, software, and telecommunications, e.g. Boeing, LG, Cisco, and Nokia [5].

CORBA describes the functionality of an object by Interface Definition Language (IDL), a host independent and declarative language. IDL is not an implementation; rather it is a specification of interfaces to object-oriented programs (methods) and has mapping standards to major programming language such as C, C++, Java, and Lisp. Therefore, IDL-specified interfaces are platform independent and can be implemented in different languages. In addition, IDL provides interface for object attributes and operations with users seeing only the IDL stubs, a set of target codes generated by an IDL compiler of objects. Hence, objects can communicate without access detailed source code.

CORBA’s central component, Object Request Broker (ORB), establishes a client-server relationship between objects and acts as a central object bus during object communication. It provides a mechanism to transparently communicate client requests to target object implementations, i.e. enabling operations on distributed objects residing anywhere on a network as if they were local objects. ORB is responsible for encompassing all communication infrastructures necessary to identify and locate objects, handle connection management, and deliver data.

For static invocations, clients communicate with the ORB Core through an IDL stub that is generated by an IDL compiler and used to represent the language mapping between the client and the ORB Core. The equivalent at the server side, called IDL skeleton, has the similar function, i.e. represent the language mapping between the object implementation and the ORB Core. All operations in the invocations are specified in advance and are known to the client and the server at compile time. For dynamic invocations, Dynamic Invocation Interface (DII) allows the client to specify requests to objects whose
definitions and interfaces are unknown until run-time. The Dynamic Skeleton Interface (DSI) provides analogous functionality for the server-side and is used to create a proxy skeleton for remote objects whose static skeletons are not bound to it. DSI typically communicates with remote ORBs.

The ORB interface provides access to every ORB service except client invocations that go through the IDL stubs or the DII, and object activations and calls. Object Adapter provides interface between objects and ORB, and is responsible for creating and interpreting object references, activating and deactivating object implementations, and invoking methods via a skeleton or DSI. The CORBA architecture is illustrated in Figure 9.

![Figure 9. The CORBA architecture](image)

CORBA provides programmers a transparent way to communicate objects in a heterogeneous, distributed environment without considering hosts, operating systems, and programming languages. CORBA was used in all the papers in this thesis to access, operate, and distribute CAD objects between similar or different operating systems.

Some work related to CORBA includes, e.g., Remote Procedure Calls (PRC), Distributed Component Object Model (DCOM), and Java Remote Method Invocation (RMI). However, despite certain progress, they are rather limited compared to CORBA:

- **RPC:**
  - Does not provide object abstractions, message passing, or dynamic invocation
  - Does not address inheritance of interfaces

- **DCOM:**
  - Platform dependent (WINDOWS)
  - Does not support heterogeneous distributed computing

- **Java RMI:**
Web Services is an emerging distributed middleware technology in addition to CORBA. The communication protocol for building Web Services is known as the Simple Object Access Protocol (SOAP). SOAP is a lightweight, XML-based, platform and language independent protocol that allows applications to exchange structured information across the Web, usually over HTTP. Compared with CORBA, Web Services/SOAP has both advantages and limitations due to some of its important characteristics (+ represents advantages, – represents limitations):

- It uses XML-encoded plain text to send data.
  + more widely adopted
  + easier for developers to read and debug
  - large message size requires much more network bandwidth
  - complex XML parsing and Marshalling needs more CPU time
- It uses HTTP protocol for communication.
  + can pass through firewalls
- Its interface is Web Services Description Language (WSDL).
  - lower level interface description language
  - more difficult to create and understand for humans (compared to CORBA/IDL) [96]
- It is a simple and lightweight protocol.
  - No standardized transaction and security context
Chapter 5. Summary of the Appended Papers

This chapter briefly summarises the appended papers concerning their causal relationship, technologies used, results, and author’s contribution.

The causal relationship between the papers is illustrated in Figure 10 where arrows mean “results in”.

Paper A addresses the integration of an MBS analysis tool and a CAD system. OO data model and query language AMOSQL are used to build the MBS model and perform MBS analysis. CORBA is identified to be the enabling technology to access and manipulate the CAD model.

Paper B uses the results of Paper A and adds distributed database primitives to achieve the closer integration addressed in Paper A. CAD functionalities are also extended by interfacing the DBMS – AMOS II.

Paper C inherits the findings of and techniques used in Paper B regarding DBMS and CORBA, and further addresses synchronisation of engineering information in collaborative design by employing active database technology.

Paper D describes architecture, configuration of the prototype system, M-Sync, and demonstrates how active DBMS can manage the consistency control issues to be solved in Paper C.

Paper E addresses the flexibility, extensibility, and maintenance issues of Papers C and D. It demonstrates how high-level database query language, peer database, and wrapper technique can be used to solve these problems and further enhance the system’s functionality.

Figure 10. Causal relationship between the papers
The technologies used in the different papers are summarized in Table 2.

<table>
<thead>
<tr>
<th>CORBA</th>
<th>Database technology</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OO data model</td>
</tr>
<tr>
<td>Paper A</td>
<td>X</td>
</tr>
<tr>
<td>Paper B</td>
<td>X</td>
</tr>
<tr>
<td>Paper C</td>
<td>X</td>
</tr>
<tr>
<td>Paper D</td>
<td>X</td>
</tr>
<tr>
<td>Paper E</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2. Technologies used in different papers

5.1 Paper A


5.1.1 Summary

This paper focuses on engineering information integration concerning orientation information of a multibody system. The methods for modelling, calculating a multibody system by means of database query language, and plotting MBS data into an MBS analysis tool are described. It further describes how to combine the data with a CAD system using its CORBA-based API. This example demonstrates an industrial MBS model being simulated in an MBS analysis tool. The orientation information of its various components is then passed over to the CAD system. The MBS analysis results, numerically represented in the MBS tool, can be combined with CAD geometry data and graphically visualized in the CAD system, as shown in Figure 11. This work shows how a commercial CAD system and an MBS tool can be integrated into an engineering analysis environment. It will also be the platform for the next step when the MBS and CAD system will be integrated more generally based on the object-relational mediation technique, as seen in Paper B.
5.1.2 Contribution

In this paper, the central enabling technologies for information integration in engineering collaborative design are primarily identified and demonstrated by integrating the orientation information of an industrial MBS model between two different systems. This paper is the basis for all other papers in the thesis.

This paper is jointly authored with R. Babelon who is responsible for modelling, simulating the MBS model, and plotting the result file, as shown in the blue rectangle in Figure 11. H. Ma is responsible for combining the MBS result file with the CAD solid model by using CORBA, as shown in the red rectangle to the right. The paper is equally written by both authors.

5.2 Paper B

Ma, H and Orsborn, K., Integration of Multibody System Analysis Information Based on Database Technology, Proc. of the 3rd International Conference on Engineering Computational Technology, Prague, Czech Republic, September 4–6 2002

5.2.1 Summary:

This paper is based on the results achieved in Paper A. Through the examples provided, this paper demonstrates how to use multi-database facilities and a high-level database query language to combine MBS data between different engineering analysis tools throughout the network as well as how to access MBS data by using a CORBA-based client-server model. Furthermore, how to use database query language to compose queries to retrieve MBS information stored in the interfaced database is also shown. A prototype system to integrate MBS data is built, as shown in Figure 12. It consists of an object-relational multi-database management system, an MBS analysis tool, and commercial MCAE software.

Figure 12. The prototype system for integration of MBS analysis data
The results of this work include:

- By using multi-database facilities and CORBA, MBS data produced by MBS analysis tools can be directly accessed through the network and combined with data from other engineering analysis tools, such as MCAE systems.
- System integration is accomplished at the query language level, i.e. users always have access to the latest MBS data since no file transfers are needed.
- Both MBS data and CAD data can be naturally stored since the extensible object-oriented data model fits with the object-oriented data structure, as found in engineering data.
- CAD functionalities are also extended by interfacing the DBMS. All data is immediately accessible through the database query language, where users have high flexibility to retrieve and combine the data originating from different representations and data sources from different domains. This is much harder to achieve in, for example, the MCAE system itself.

5.2.2 Contribution

This paper partially answers the first two research questions and further demonstrates the capabilities of DBMS and CORBA with respect to information integration, data representation and management, and extensibility for functionalities of traditional CAD system.

H. Ma initialised the main idea and did the implementation work. Both authors contributed equally to the writing.

5.3 Paper C

Ma, H., Johansson, H., and Orsborn, K., Distribution and Synchronisation of Engineering Information Using Active Database Technology, preliminarily accepted by the Journal of Advances of Engineering Software. (Note: this is revised version of the original paper that has been published at the 7th International Conference on the Application of Artificial Intelligence to Civil and Structural, Egmond aan Zee, The Netherlands, 2-4 September 2003)

5.3.1 Summary

Team members involved in distributed engineering collaborative work need to be able to share and exchange design information and ideas. This work presents an active database approach to enable this prompt exchange of engineering information among distributed team members. A prototype system, M-Sync, is constructed to support mechanical computer aided engineering (MCAE), where co-located engineers engaged in the same engineering design work can interactively and simultaneously work on the same solid model. By interfacing a database management system supporting active database rules and peer-to-peer communication, information sharing among team members and MCAE systems
can be supported. Only updates made in one MCAE system are actively identified and distributed to other peer MCAE systems, i.e. the information residing in the distributed MCAE systems is automatically synchronised. A simple locking mechanism is also implemented to resolve user-level transaction problems, i.e. avoiding conflicts caused by concurrent synchronisations issued by multi-users.

The architecture of the prototyping system is shown in Figure 13.

![Figure 13. The M-Sync system architecture](image)

The results of the current approach include the following:

This active database technology provides mechanisms to monitor, transform, locate, and query data between distributed engineering applications, resulting in more powerful methods for sharing engineering information. Together with built-in communication primitives of the DBMS, a distributed P2P engineering information system is supported.

This approach ensures that all team members have up-to-date and consistent information, as database rules actively monitor the data and database mediators immediately respond to inter-database requests. It can also improve the functionality of distributed engineering applications, since each team member can have full access to all information locally. Compared to approaches where the complete model is distributed at updates, this model reduces the necessary data transfer during updates by sending only updated data to other team members.

The general capabilities of the DBMS, such as storage management, data modelling, query language, query processing, transactions, and meta-data, can provide easier and more efficient data management compared to equivalent implementations in conventional programming languages.
5.3.2 Contribution

This paper addresses all research questions and demonstrates how advanced database features can considerably assist engineering collaborative design in terms of information synchronisation and distribution.

H. Johansson and H. Ma proposed the design idea, while H. Ma did the implementation work. H. Ma wrote the main part of the paper with valuable guidance from Orsborn and discussions with Johansson.

5.4 Paper D

Ma, H., M-Sync: A System for Distributed Collaborative Design, submitted for journal publication

5.4.1 Summary

This work employs database technology and distributed computing to the area of engineering collaborative design. The focus is to study some of the important technologies and architectures that can facilitate distributed collaborative design using conventional CAD systems. Through the development of the M-Sync prototype system, it is shown that database technology can provide important functionalities that are required in a collaborative design system using conventional CAD systems. In this work, the database management system not only provides direct support for the effective storage of and efficient access to complex CAD data, but also assists collaborative design activities by offering advanced database features such as an extensible object-oriented query language, active rules, and multi-database facilities. Specifically, the DBMS is used to identify and propagate data updates made by the CAD system, which is a critical issue in engineering collaborative design. It further maintains data consistency between the CAD system and the database which is another critical issue when data is distributed and replicated. This work also employs the distributed object computing to access, operate, and distribute CAD system objects between the CAD system and the DBMS in a platform independent manner.

The system architecture and basic transaction management are also studied. The present work extends functionalities of the stand-alone and single-user CAD system and facilitates synchronous collaborative design where distributed designers perform online interactive discussions regarding incremental modifications of a CAD system model. The proposed approach is implemented as a collaborative CAD system environment where distributed design information can be actively synchronised between co-located designers when typical collaborative design tasks such as solid and assembly modelling are being performed. The service models in the prototype system are illustrated in Figure 14.
5.4.2 Contribution

This further develops the M-Sync system proposed in Paper C. This system can manage basic data consistency between the CAD system and the DBMS as well as higher level manipulation on a CAD solid model.

5.5 Paper E

Ma, H. and Risch, T., A Database Approach for Information Communication in a Peer-to-Peer Collaborative CAD Environment, submitted for journal publication

5.5.1 Summary

Timely and efficient information communication is a key factor in ensuring successful collaboration in engineering collaborative design. This work proposes a database approach to support information communication between distributed and autonomous CAD systems. It provides the designer an easy and flexible way, a project-based propagation meta-table, to specify what parts of a CAD information model should be communicated to other collaborating designers. A CAD peer manager, containing a peer database that stores information to be exchanged with the other collaborators, wraps each participating CAD system. The peer manager identifies changes made to the CAD model by downloading data from the CAD to the peer database. The changes are detected by stored procedures and active rules in the peer database that are automatically generated based on the propagation meta-table. The updates of the types of data that the user has specified to be shared between peers are timely propagated to other
peers via inter-database message passing, thereby minimizing the volume of necessary information to be exchanged. When designers receive information concerning the updates from other peers they can freely incorporate, filter, or delete the received updates by manipulating corresponding messages using a *propagation control interface*, which is also used to issue user’s commands to download the data from the CAD system to the peer database and lookup the received messages in the peer database. The approach is applicable on any CAD system having a CORBA interface and can be applied also to other kinds of object-oriented interfaces. The flowcharts of message manipulation in the propagation control interface are shown in Figure 15.

![Flowcharts](image.png)

**Figure 15.** The flowcharts for (a) message lookup, (b) uploading data to CAD, and (c) CAD data downloading

### 5.5.2 Contribution

This paper addresses the flexibility, extensibility, and maintenance issues of Papers C and D. It demonstrates that active mediator systems with P2P architecture can be used to synchronise CAD systems, and that it is possible to automatically generate the wrappers and the active rules from the project-based propagation meta-table containing descriptions of which part of logical CAD system data should be exchanged between the peers.

The idea was proposed by both authors. H. Ma did the main part of the implementation work and wrote the paper with valuable guidance from T. Risch.
Chapter 6. Conclusions and Future Work

This thesis points out the characteristics and limitations of current computer-based tools in terms of functionalities for online engineering collaborative design. Requirements for future information systems supporting integrated collaborative design as well as possible solutions meeting these requirements have been identified and discussed. Various technologies and methodologies, such as groupware in Computer-Supported Cooperative Work, product modelling (using VRML, XML or Java), user interaction, system architecture, and industry standards for data exchange, have been investigated.

Modern database technology and CORBA were selected here as the principal enabling technologies to facilitate the integration and communication of engineering information in collaborative design. The proposed methods using these technologies were implemented and validated by constructing several prototype systems, as presented in the five research papers. Each paper focuses on specific problems and contributes findings to the research questions and overall goal of the work.

Major conclusions drawn from this work are the following:

- Database technology facilitates the information management and interactive work collaboration between users by offering unique database features such as an extensible object-oriented data model, a powerful declarative query language, active database rules, peer database architecture, and a wrapper-mediator approach. Generally, it is believed that database technology will play a very important role concerning information management for distributed engineering collaborative design, as it is currently doing in traditional business and administrative applications. More specifically,
  - An object-relational database supports the object-oriented data model and a high-level declarative query language. These features can minimize information loss in terms of engineering data representation and maximize the level of expression and the capability to retrieve and manipulate data.
  - Engineering collaborative design requires a shared or autonomous workspace, and preferably with the full capability to perform design tasks on all levels. A promising way to achieve this is a peer-to-peer architecture where the wrapper-mediator approach is used to wrap
the distributed and autonomous design tools to support information integration between these systems. Active rules are used to capture logical changes in CAD data while P2P facilities are used to communicate the changes with other wrapped CAD peers.

- Real-time information communication and user interaction can be supported by active database rules. For instance, active rules can timely identify updates introduced by designers and automatically propagate these updates to others.

- CORBA provides an object-oriented, uniform, and independent interface to describe object functionality. CORBA relieves users' work by hiding the implementation detail of an object and providing users a high-level and transparent way to communicate objects in a heterogeneous, distributed environment without considering hosts, operating systems, and programming languages. CORBA is used in all the papers of this thesis to create, access, operate, and distribute CAD objects, realising information integration in a platform, hardware, and software-independent manner.

- Compared with employing a database management system and a CORBA specification to implement the above features, a conventional programming language works on a lower level in terms of programming efforts, reuse of system functionality, and system scalability, complexity, and maintenance.

### 6.1 Future work

Several research issues can be identified as future work:

More complete mechanisms need to be developed for the current prototype systems, thereby addressing complicated dynamic issues caused by the distributive and autonomous working manner, including versioning, transaction management, and recovery. Furthermore, although the current M-Sync system can automatically identify and propagate updates of a CAD model, the downloading and uploading of CAD data are static. To make it fully automatic, it is necessary to develop mappings for types and functions between CORBA and DBMS. It should also be interesting and valuable to investigate and evaluate similar work on accessing wrapped CAD systems from mediators [97].

The development of a database mediator with a STEP-based wrapper could be an interesting research in the long run [Koparanova and Risch, 2002]. Successful information systems supporting distributed engineering collaborative design must have sufficient capabilities to both integrate and process data from distributed and heterogeneous data sources. The STEP standard provides widely-accepted and mature data representation for heterogeneous engineering data to be integrated, while a database wrapper-mediator approach, together with a powerful query language, provides the means to efficiently access, combine, and process data. Therefore, by combining these two complementary technologies, CAD systems in the long run can be served as standardised plugins in engineering collaborative design.
Integration and Communication of Engineering Information in Collaborative Design

References

5. CORBA, http://www.corba.org/


43. Huang, K., Li, X., Cao, S., Yang, B., and Pan, W., Co-DARFAD - The collaborative mechanical product design system, *Proc. of the International


Paper A
Integration of a multibody system analysis tool and a CAD system

Haoxue Ma\textsuperscript{1} and Romain Babelon\textsuperscript{2}

Division of Computer Aided Design\textsuperscript{1}
Luleå University of Technology, SE-971 87, Luleå
Department of Machine Design\textsuperscript{2}
Royal Institute of Technology, SE-100 44, Stockholm

Abstract

The work presented in this paper is the first step in the integration of a Multibody System Analysis (MSA) tool and a CAD system. Through the practical example described, a multibody system from industry, it is shown how the MSA data has been produced in a MSA tool and how the data has been imported to a CAD system via the CAD API. The analysis data that is represented numerically in the MSA tool can therefore be examined graphically in the CAD system. This work shows how a commercial CAD system and a MSA tool can be integrated in an engineering analysis environment, and will be the platform for the next step where the MSA tool and the CAD system will be integrated in a more general way on the basis of the object-relational mediation technique. In the prototype system developed in this work, the CAD system I-DEAS is used as the tool for solid modelling and animation, and MECHAMOS is used for analysis and preparation of MSA data.

Keywords
integration, CAD, MSA, API, Multibody System

1. Introduction

With the rapid development of computer-based engineering design and analysis environments, more and more complex engineering problems can be solved using advanced computer aided tools such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Finite Element Analysis (FEA) and Multibody System Analysis
(MSA). Often, several computer tools are used together to deal with one practical engineering problem. These tools can be used successfully one by one, but the great benefits are gained through system integration, where data created or changed in one application can immediately be used in the other steps of the development process [1]. For example, FEA may need to combine geometrical data from a CAD model and loads from a MSA model to perform an analysis of a structure.

The role of MSA in product development is becoming more and more important. To handle large and complex MSA models, many analysis tools have been developed. When designing a mechanical system, the dynamic behaviour of a product is one of the main concerns. It is therefore important that the simulation tools used for the analysis of dynamic behaviour of a product in different disciplines can be integrated.

Application-specific integration is one way to integrate systems, see Figure 1a. However, a large number of interfaces need to be written when many applications are involved. Interfaces also need to be replaced for each new version of an application, and the required number of interfaces grows exponentially with the number of applications [2]. By using a central database, see Figure 1b, the required number of interfaces can be significantly reduced.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{Integrated system and the main area of interest of this work}
\end{figure}

This paper focuses on engineering data integration. More precisely by using the CAD API, MSA information from a MSA tool can be imported into a CAD system. The MSA data is produced by a simulation run in the MSA tool. The motions of bodies that are represented numerically in
the MSA tool can be visualized in the CAD system. This is a good way to examine MSA data, to visually check problems such as incorrect key dimensions of MSA components, unexpected kinematic trajectories and possible interferences between components, etc. This work is the first step in the integration of these two systems. The final goal is a tight integration of the two systems where data can be requested and accessed entirely through object-relational query language.

2. Requirements for the systems to be integrated

In order to fulfil the final goal of this work – a tight integration of the two systems - the systems to be integrated should have the following characteristics:

2.1 CAD system

There are a large number of CAD systems available that engineers can take advantage of to deal with different kinds of engineering problems. Some CAD systems focus mainly on 2D drawing whilst others consist of a number of software modules supporting complicated engineering activities such as solid modelling, FEM, CFD, etc. CAD systems suitable for this work should include features like the following:

- Solid modelling. This is a prerequisite for computer integrated manufacturing [2], as solid models can be used by many downstream activities. In the case of integration of MSA tools and CAD systems, solid modelling provides the necessary information, e.g. mass and inertia properties, required by the MSA tools. It also provides the geometrical information about components needed for animation.

- Assembly modelling. An assembly is a modelling entity that combines individual solids into a logical hierarchy that relates to the spatial packaging of the physical assembly itself. In this work assemblies are used to visualize the complete assembly of solid models, to calculate interferences and unexpected kinematic trajectories.
CAD API. In order to integrate with MSA tools, CAD systems should have an application programming interface (API) to both access and manipulate the CAD database, where a complete set of data required by the MSA tools is stored, and to communicate with other external applications.

2.2 Multibody system analysis tool

Clearly the MSA tool used in this project should have the basic functionality required for multibody system analysis. The tool should also give the possibility to manage assemblies, to perform numerical integration of the solids’ motion, and to enable the numerical results to be visualised using some plotting facilities.

The MSA tool should also enable users and developers to easily access and retrieve assembly data and computational results. The reason for this is that the easier the data is to retrieve and to put in a neutral format, the easier it is to share it with another software.

3. The systems chosen for this work

The systems chosen for this work are the commercial CAD system I-DEAS [3] and MECHAMOS [4] - a MSA tool based on an object-relational database management system. The CAD system is used as the tool for solid modelling, assembly modelling and animation. The MSA tool is used for analysis and preparation of MSA data.

3.1 I-DEAS

I-DEAS, which is developed by SDRC (Structural Dynamics Research Corporation), is one of the leading CAD systems for mechanical design on the market. It is composed of a number of software modules that can be used to deal with various engineering applications, e.g. solid modelling, assembly modelling, tolerance analysis, FEM simulation, etc.

Open I-DEAS (OI) [5], the I-DEAS API, is one of the I-DEAS Open Architecture components. It provides users with C++ based routines to directly access the I-DEAS database. With OI, geometry and simulation
data in I-DEAS can be directly accessed and user-defined attributes can be associated with I-DEAS entities. OI enables users to create applications that communicate with I-DEAS section via Orbix, which is an implementation of the CORBA [6] standard.

All these features provide I-DEAS with the possibility of customizing, extending its capabilities and tightly integrating with other application tools.

3.2 MECHAMOS

MECHAMOS is a MSA tool based on the object-relational database management system (ORDBMS) AMOS II [7]. Object-relational database technology enables the users to define object-oriented data structures and to manipulate and retrieve the data using advanced query facilities. In MECHAMOS all the data concerning the assembly models is fully and easily accessible through the query language and can be put into a suitable form for further analysis.

MECHAMOS has actually been created by the combination and integration of three different tools:

- The database management system AMOS II: an object-oriented structure is defined in an AMOS database.

- The symbolic mathematical software MAPLE, and especially the SOPHIA [8] package of functions for MSA.

- The numerical mathematical software MATLAB.

The database structure manages the structures of mechanisms, symbolic equations defining the system’s behaviour and numerical sequences resulting from the numerical integration of the movement.

MAPLE is used to produce the equations of motion of the dynamic systems. These equations are produced from a set of elementary equations entered by the user: the number of degrees of freedom in the system, the relative position of the different bodies, the forces applied on the system, the masses, and the moments of inertia, etc.
MATLAB is mainly used for the numerical integration of the equations of motion produced by MAPLE and to display the numerical results (plotting facilities).

The main idea behind MECHAMOS is that simulation is initiated by database queries. The query language used is AMOSQL, a variant of traditional OSQL [9]. AMOSQL was specifically developed for object-relational systems.

Once a mechanical model is defined in MECHAMOS, the user of the package only needs to make queries to retrieve information concerning the system’s behaviour. If the information asked for cannot be found in the database, i.e. it has not been computed yet, it will be automatically calculated by the system. For example the following query:

```
SELECT applot_o("omega", max(S), N)
FROM MSA_numeric N, scalar_sequence S
WHERE name(model_oid(N)) = "Sliding Pendulum"
AND S = numseq("q1", N);
```

is made by a user who wants to see the effect of a parameter ‘omega’ on a coordinate ‘q1’, while studying a model named ‘Sliding Pendulum’. Thus, user asks the system to plot, for each value of ‘omega’, the maximum value reached by ‘q1’ during a movement.

What will actually happen in the system is that it will produce the equations of motion of the model named Sliding Pendulum. These equations will then be integrated for all the different parameter values for omega defined in the database. For each of these simulations the system will extract the maximum value reached by the generalized coordinate q1 during the integration. The results will eventually be displayed in a plot. In this example, one single query launches several simulations.

4. Integration of the MSA tool and the CAD system

The most common way of linking applications together is by interfacing [10]. This paper presents an entity-level interfacing. The following sections describe the procedure for interfacing the two systems.
4.1 Preparation of the MSA data

The whole process of data preparation is rather straightforward, as is shown below:

- A classical integration of the equations of motion for a certain numerical set of parameters, e.g. integration time, precision of the integrator, numerical values of the various parameters. This integration returns the values of the general coordinate describing the system (in MECHAMOS the MSA system state is described by a set of general coordinates).

- The MATLAB integration solver that is used to obtain numerical results changes its time step throughout the integration process. Since the CAD tools, including I-DEAS, replay sequences with a constant frame rate, numerical values need to be calculated with a fixed time interval. Thus the MATLAB results are interpolated to obtain a fixed time interval.

- The I-DEAS API conditions the data describing the state of the assembly to be sent under the format of transformation matrices (i.e. rotation + translation) for each body of the assembly. Thus, the symbolic equations of the position vectors and rotation matrices are extracted from MECHAMOS.

- Finally the transformation matrices are calculated with a fixed step and written to a file.

The process described here is also shown in Figure 2. The process is launched from within MECHAMOS, and one single MECHAMOS function performs all these operations.

4.2 Simulation of the MSA data

The procedure for simulation of the MSA data is shown in Figure 3.
Figure 2. Procedure for creating the animation file
- Process initiated by the derived function:
  animate (mbs_numeric NUM)
4.2.1 Build solid models

In order to perform the simulation, the solid models of the assembly need to be built in a CAD system. To simplify the designer’s task, the outlines of the solid models of the CAD system and the MSA tool can be slightly different. However, the geometric relations among the model joints, e.g. the relative positions of the various joints of the components, must be identical.

4.2.2 Define assembly

In I-DEAS each assembly defines, or embodies, two important modelling relationships: the spatial configuration (orientation) of instances and the non-spatial hierarchy of subassemblies and parts. The instances of solid models need to be added to the hierarchical structure. The information about the spatial configuration of instances can be automatically stored in the I-DEAS database, while MSA data is imported to I-DEAS using OI.

4.2.3 Data importing

A program based on C++ and OI routines is the interface between MECHAMOS and I-DEAS. User-defined attributes, in this case MSA data, can be tightly integrated with I-DEAS entities, in this case the
instances of solid models. The program populates the I-DEAS database with configurations, i.e. the spatial orientation of instances. The new data is created on the basis of the MSA data. I-DEAS users can then use the standard functions to visualize the animation, which is based on a set of configurations.

5. Example – lifting subsystem of a wheel loader

The lifting subsystem of a wheel loader, see Figure 4, is the test case presented in this paper. The lifting subsystem is composed of a bucket supported by two identical articulated arms. Two hydraulic pistons govern the movements of the arms. The arms’ components are connected by joints.

Figure 5 shows a side view of the mechanism. Only one of the two articulated arms is represented. The named points represent the location of the joints connecting the solid components. It can be noted that the three points O, P and E are some interface points between the lifting system and the main chassis of the loader. These points were considered grounded in this study. For the simulation a half-system was modelled: it consists of a 2D model of the system with a single lifting arm. Another approximation is that the bodies are stiff.

Eight generalized coordinates define the state of the system. They are the following: 2 translations and 6 rotations corresponding to the translations of the 2 pistons and to the rotations at the 6 joints. These coordinates describe the system completely since the position vectors and rotation angles of all components can be derived from them.

However, these coordinates (q1, ... , q8) are redundant, since the system state can be totally defined by the two pistons’ positions. All the eight generalized coordinates are used, instead of only two, to describe the position of the bodies because the equations turn out to be simpler. It is easier for a user to initiate the system in this way. Still, the only variables are the derivatives of two coordinates when deriving these eight generalized coordinates. The other derivatives can be automatically calculated using the first two coordinates.
The system state is represented by a state vector \((u_1, u_2, q_1, \ldots, q_8)\), \(u_1\) and \(u_2\) being the derivatives of the coordinates \(q_1\) and \(q_2\), respectively. The result obtained after mathematical treatment and simulation in MECHAMOS is a series of state vectors defining the system state at different time steps.

The procedure ‘animate’, shown previously, was used to export the simulation results. The result was a file with the transformation matrices of the different solids of the system at different times.
The solid models of the components are built in I-DEAS. As discussed in Section 4.2.1 the geometric relationships of the I-DEAS model joints match those of the MSA models in order to precisely match the orientation of each component. The assembly is defined and all instances of the solid models are put into the hierarchical structure of the assembly in I-DEAS.

By running the interface program, the bodies’ orientation information described in the MSA data file is transferred to I-DEAS, and configurations and animation are created.

There are two CAD constraint-related issues that need to be clarified:

- The way in which constraints are used in CAD systems is not always suitable for MSA tools. It should be noted that it is difficult to constrain a large assembly in a complete and unambiguous way in CAD systems, e.g. when designing an assembly it is not uncommon that the assemblies or their subassemblies are over-constrained or under-constrained. Currently it is not easy to properly apply CAD constraints to MSA tools, as ‘ill-constrained’ assemblies could cause the data of MSA motions to be calculated incorrectly.

- When the data in the MSA file is imported into a CAD system, the instances will be moved one by one to the positions defined in the MSA result file. However, movement of individual instance will conflict with the current spatial configuration of the instance defined by its assembly constraints. If the assembly constraints are active, each instance will be re-located again by its assembly constraints and the behaviours of the instances may be unpredictable. For this reason, the assembly constraints are redundant and should be ‘ignored’ or ‘unused’ while the MSA data is imported into the CAD system.

6. Conclusions and discussion

This paper describes the first step in the integration of a Multibody System Analysis tool and a CAD system, and clarifies CAD constraint-related issues that need to be addressed during the integration. Through
the example described – the lifting subsystem of a wheel loader – it is shown how the MSA data is produced in a MSA tool and how the data is imported into a CAD system via the CAD API. In the particular case studied it gives the possibilities to:

- visualize in terms of motions of solid models, the numerical results from the multibody system analysis

- share more accurate pre-analysed MSA information from other MSA tools with CAD

- be the foundation for the next step, where the MSA and the CAD system will be integrated in a more general way on the basis of the object-relational mediation technique

The next step in the integration of the two systems will be to bi-directionally transfer more complex information without the support of any intermediate file. To fulfil these requirements, the object-relational mediation technique should be made available to the CAD system; i.e. by means of AMOSQL queries CAD users should have the possibility to directly read and manipulate the MECHAMOS database and CAD data queried directly from MECHAMOS, see Figure 6.

![Figure 6. Integration of MECHAMOS and CAD](image)

In this scenario, both applications will share data, since they can directly store and extract information from the same set of databases. The ORDBMS and its query language will make it possible for
non-expert database programmers to enjoy the benefits of object oriented databases.

References:


Integration of Multibody System Analysis Information Based on Database Technology

Haoxue Ma* and Kjell Orsborn+

* Department of Mechanical Engineering, Luleå University of Technology, Sweden
+ Department of Information Science, Uppsala University, Sweden

Abstract

The integration of data from distributed and heterogeneous data sources is an important issue in the product development process. This paper shows how modern database technology can be used to facilitate the integration of engineering information, specifically concerning multibody system (MBS) analysis information, in this process. Through the example provided, it shows how to use an object-relational multi-database management system as an embedded database to integrate and manage MBS analysis information. It is further shown how to obtain MBS analysis information from an engineering data source over the Internet using a middleware-based client-server model. By integrating an object-relational multi-database management system, engineering applications not only gain general database capabilities such as, storage management, query language, query processing, and transactions, but also the possibility of combining data from different representations and data sources.

Keywords: multibody system, data integration, query language, object-relational multi-database management system, embedded database, distributed and heterogeneous, data source, middleware.

1 Introduction

Computer-based tools and methods have become important means of design, simulation, and manufacturing in the product development process. Today, many companies function as global enterprises where nearly all the work is distributed and located worldwide. Consequently large amounts of heterogeneous and distributed engineering data are produced. Furthermore, Computer-based analysis activities within different analysis domains normally need to combine, exchange, and share relevant product data to each other. For instance, finite element analysis (FEA) and
computational fluid dynamics (CFD) may need to use geometrical data from a solid model and loads from a multibody system to perform structural analysis and fluid analysis. This trend is ever increasing. Product development therefore requires product information to be communicated and integrated between computer-based analysis tools in an engineering information system (EIS) consisting of a globally distributed and heterogeneous computing environment.

Relational database technology has been in the market for about twenty years having a very strong market share. This success is partly gained by the support of the standardized query language SQL. Relational database management systems (RDBMSs) were originally developed for handling administrative and business information but have also been applied for managing product data [1]. However, in comparison to these traditional administrative database applications, scientific and engineering applications usually involve models and analysis methods of higher complexity. These applications have requirements and characteristics that differ from those of traditional administrative database applications, such as more complex structures for objects, longer-duration transactions, new data types for storing images. This indicates that traditional database management systems (DBMSs) are unsuitable to represent and handle these kinds of data. One way to efficiently manage engineering data is to use object-relational database management system where object-oriented approach offers the flexibility to handle some of these requirements without being limited by the data types and query languages available in traditional database systems [1, 2].

Data that need to be integrated are normally distributed and located worldwide on various computing platforms. This makes it difficult to integrate data between different engineering applications. Software that provides a link between disparate applications running on different hosts connected to a network is called middleware. Middleware provides services for developing client/server applications and communication across different platforms. Using middleware approach to realize the integration of engineering information is one of the desirable ways. Hence, users can access and integrate engineering data via networks regardless of the hardware or software being used.

Related works have shown that database technology could be a key technology to support engineering data integration [3, 4, 5]. However, DBMSs in these works are mainly used as persistent storage for a specific application, to store and access relational data. More advanced operations to use a query language to query, combine, filter, and exchange data between distributed DBMSs are not available.

This paper presents how modern database technology can be used to facilitate the integration of engineering data and applications. More specifically, the application domain concerns information integration in multibody system analysis. By integrating database technology, engineering applications gain general database capabilities such as storage management, uniform data model, meta data, query language, query processing, and transactions. The technology further support a
query-language-based access to remote data, that is a key element to provide sharing, exchange and combination of data in a distributed and heterogeneous EIS. It is further shown how to obtain MBS analysis information from an engineering data source over the Internet using a middleware-based client-server model.

2 Integration of Engineering Information

A prototyping system using the multi-database management system architecture is being developed for integrating MBS analysis information, as shown in Figure 1. The system consists of the object-relational database management system AMOS II [6], MBS analysis tool MECHAMOS [7], the Mechanical Computer Aided Engineering (MCAE) system I-DEAS [8], and GOOVI [9] - a general object-oriented graphical browser for the AMOS II database. Here, AMOS II is embedded within I-DEAS via MyApplication– an in-house developed interface program written in C++. The intention to develop this system is to find out how to integrate engineering information, specifically concerning MBS analysis information, using an ORDBMS. Details of the system are explained in section 3.

![Figure 1. The scenario of an integrated engineering information system using the AMOSII multi-database management system architecture](image)

2.1 Earlier Work and Scenario of the Prototyping System

Earlier work [10] has shown that by using I-DEAS application programming interface (API), the MBS analysis orientation information produced by MECHAMOS could be imported into I-DEAS. The orientation information was presented in a numerical form and saved in a text file. It consists of 13 columns, i.e. 1 column represents the time step, and the other 12 columns represent the values of the 4*3 transformation matrix of each body. Therefore, the bodies’ motions that are presented in a numerical form could be visualized in I-DEAS with the multi-bodies’ movements.

While in this integrated engineering information system, it is possible for MECHAMOS, I-DEAS to directly share and exchange their data between each other at query language level, where no file transfers are needed. For instance,
MECHAMOS uses I-DEAS geometrical and topological data of solid models as input to build mathematical models then I-DEAS send queries directly to MECHAMOS to retrieve information concerning the MBS behaviour and visualizes the MBS motions. GOOVI can be used to view, query, and update their data schemas, instances, and functions in this federation. In general by using this approach, engineering applications within other analysis domains could also have possibility to combine and share data between each other to perform domain-specific analysis.

2.2 AMOS II Database Management System

The AMOS II DBMS, developed at Uppsala University, Sweden, is a main-memory resident and object-relational DBMS. Furthermore it is equipped with multi-database management facilities making it possible to access and combine data from several distributed data sources through a collection of AMOS II servers. AMOS II is built upon an extensible object-oriented data model. It makes it simpler and more feasible to handle complex data structures, such as data found in MBS analysis. The basic concepts of the data model are types, functions, and objects corresponding to classes, methods, and instances in the object-oriented programming language. The data model is accessible through AMOSQL, an object-oriented query language, which is used to define, populate, query, and update the database. AMOS II can further be embedded within other systems by using either of its C, Java, or LISP interfaces so that applications can take advantage of database facilities such as storage management, uniform data model, meta data, query language, query processing, and transactions.

The AMOS II multi-database facilities allow several AMOS II servers to connect and communicate over a network where each AMOS II server also is a DBMS of its own [11, 12], a simplified architecture is shown in Figure 2.

![Figure 2. Simplified AMOS II multi-database management system architecture](image-url)
This feature is a base for this work since it provides an architecture and functionality for engineering data to be integrated over a distributed and heterogeneous computing environment. As illustrated in Figure 2, application program can access and combine data from distributed data sources via a group of AMOS II servers, such as embedded AMOS II, AMOS II servers, and name server. The data sources can for example be a RDBMS, a text file, or another AMOS II server. In this federation, AMOS II servers manage and integrate inter-database requests and data from other servers. A name server which is also an AMOS II server manages these AMOS II servers. The name server keeps track of the servers since its local database contains the names, locations, and other general data regarding the servers. External data is accessed, translated, and combined via AMOS II translators or wrappers [13]. Therefore, data that is in different representations and located at different data sources can be freely extracted, combined, and filtered at the query language level.

2.3 The MECHAMOS System

MECHAMOS is a MBS analysis tool developed at The Royal Institute of Technology, Sweden. It is the combined package of AMOS II, MapleV, and Matlab. In MECHAMOS, MapleV and Matlab enable the system to perform MBS analysis in both numerical and symbolical ways. All the data concerning the MBS assembly models are fully and easily accessible through the AMOSQL and can be put into a suitable form for further analysis. Once a mathematical model of MBS is defined in MECHAMOS, the user only needs to make certain queries to retrieve information concerning the system’s behaviour. If the information asked for cannot be found in the database, i.e. it has not yet been computed, it will be automatically calculated by MECHAMOS.

2.4 CORBA Standard and MCAE software I-DEAS

The common object request broker architecture (CORBA) [14] is one of the most commonly used middleware. It was specified by The Object Management Group (OMG) to promote the theory and practice of object technology for developing distributed computing systems. CORBA is an architecture and specification for creating, distributing, and managing distributed program objects in a network. It requires all objects to be expressed in its Interface Definition Language (IDL). User can only see the IDL stubs, which is a set of target codes generated by IDL compiler, of objects therefore objects can communicate without access detailed source code behind. CORBA allows programs running on different platforms at different locations and developed by different vendors to communicate over the network through its central component, the object request broker (ORB). An ORB encompasses all the communication infrastructures necessary to identify and locate objects, handle connection management, and deliver data.
Based on a shallow evaluation, we think there are two main differences between distribution by using CORBA and AMOS II multi-database facility. The first one is that CORBA is originally developed for objects representing and communicating between different computing platforms rather than data processing. There are not database-facilities-like features exist in CORBA, such as indexing. When comes to complex problems, it may not as efficient as AMOS II to process data. The other difference is that AMOS II uses TCP/IP as the inter-process communication transport mechanism and multi-database query optimiser to speed up data processing. CORBA uses Internet Inter-ORB Protocol (IIOP) to provide interoperability with other compatible ORBs. The IIOP is based on TCP/IP but some additional layers are defined to transfer CORBA requests between ORBs. It is somewhat slower when large amounts of data need to transfer.

I-DEAS is one of the major MCAE systems. It supports engineering activities such as solid modelling, finite-element simulation, manufacturing simulation, etc. Open I-DEAS (OI) is a flexible open architecture toolkit used for customizing, automating, and integrating custom and third party applications with I-DEAS [15]. Via Open I-DEAS, an external application can access all I-DEAS commands and functions as well as internal data representations, including all generated information such as geometrical and topological data, finite element analysis results. Open I-DEAS is also an object-oriented client-server implementation. It uses a product called Orbix, which is an ORB and complies with CORBA standard, from IONA technologies as the base technology. Therefore by using Open I-DEAS, I-DEAS data can be closely integrated with external applications without considering platforms and hosts.

3. An Integrated Engineering Information System for MBS Analysis

The current work is shown in Figure 3. The aim of the current work is to realize a part of central works of the above scenario, i.e. by using AMOS II multi-database facilities, embedded feature, and Orbix-based client-server model to

- access MECHAMOS MBS orientation information and transfer them to I-DEAS, then combine I-DEAS solid models to visualize the multibody system motions
- access and query I-DEAS MBS data from the embedded AMOS II
- view and manage the MBS data schema with GOOVI

In this system, the interface program MyApplication connects local Orbix via IDL stub and links AMOS II via its C API. On the server side, IDL skeleton, target code produced by the IDL compiler at server side, links I-DEAS and Orbix together. Requests called from Client side are transferred via network and handled by Orbix.
Hence, AMOS II is embedded within I-DEAS via AMOS II C interface and Orbix supported by Open I-DEAS. The MECHAMOS MBS orientation data are populated and stored in a remote AMOS II server call AMOS II*. Using AMOS II multi-database facilities, these MBS data are accessed from the embedded AMOS II then transferred to and visualised in I-DEAS via Orbix and Open I-DEAS functions. The detailed description of the system is explained as the following.

![Diagram](image)

**Figure 3.** The Prototyping system for integrating MBS analysis data where the MECHAMOS MBS orientation data is populated in AMOS II*

### 3.1 Accessing I-DEAS Using Orbix

As described above, external application programs access I-DEAS commands and internal data by using Orbix. The architecture of accessing I-DEAS data is illustrated in Figure 4. This is a typical CORBA architecture.

MyApplication and the embedded AMOS II act as a client application in the client-server model. The client application makes the program call, e.g. call method to get the inertia values of certain parts, to the client IDL stub. Stub packs the code and sends it over the network via Orbix to the remote Orbix server that starts concurrently at I-DEAS start. The server skeleton unpacks the code and makes the appropriate program call. The Orbix object - OI_Server, which is created at the Orbix server start up and is also a gateway for other objects, executes the program call. OI_Server calls other relevant secondary Open I-DEAS servers and inherited methods, e.g. command server and GUI server, to perform the remote calls that are invoked from the client application. Once the program call is completed at the server side, all output and return values are packed and sent back to the client via Orbix. The stub unpacks these values and returns them to the client, just as if the client had made the program call locally. In the prototyping system, the interface program
MyApplication is compiled and runs on Windows 2000, and I-DEAS runs on a UNIX workstation.

![Architecture of accessing I-DEAS using Orbix](image)

**Figure 4.** Architecture of accessing I-DEAS using Orbix

### 3.2 Accessing MECHAMOS MBS Orientation Information

The requests of accessing the MBS data that stored in the AMOS II* are sent from MyApplication. They are processed and transferred to AMOS II* through the embedded AMOS II. There are two ways to communicate with AMOS II from the host language of the application: either by using the **embedded query interface** or the **fast-path interface**. In the embedded query interface, AMOSQL statements can be executed from the application by calling an AMOSQL execution procedure that takes strings of AMOSQL statements as input. After evaluating the statement, AMOS II returns the results. This interface is slower than the fast-path interface since the AMOSQL statements are parsed, optimised and compiled on-the-fly. In the fast-path interface, predefined and precompiled AMOS II functions can be called from the application without the overhead of query processing. This means it can be much faster than the embedded query interface, especially when the same expression needs to be executed repeatedly. In the work both interfaces are used to communicate with AMOS II and populate MBS analysis data. Fast-path interfaces are utilised on the calls that are executed repeatedly to increase execution speed. Embedded query interfaces are used to the calls with simple process and only need to be executed once.

Suppose an AMSO II name server `MBS_NameServer` is running on a host called “myhost.luth.se”, the AMOS II* can be registered into the multi-database federation by doing the following:

1. start an AMOS II system where MBS orientation data are populated
2. call the AMOSQL function: `register('MBS_DB', 'myhost.luth.se');`
3. start listening for inter-database requests from other AMOS II systems by calling the AMOSQL function: `listen();`

From now, this AMOS II is registered in the federation through the name server `MBS_NameServer`. It serves as a remote AMOS II server called “MBS_DB” where MBS orientation information is stored. It corresponds to the AMOS II* shown in Figure 3. The following code snippet shows how to access MBS data stored in AMOS II* by calling the embedded AMOS II from `MyApplication`.

```c
a_execute (connection, scan, "register ('Embedded_DB',
              'myhost.luth.se');", FALSE);

a_execute (connection, scan, "ship('MBS_DB', 'select t from
transform t;')", FALSE);
```

In the above code, `a_execute` is a function of AMOS II C interface to pass the AMOSQL statement that is between the double quotation marks to AMOS II for evaluation, `connection` is a connection handle, `scan` is a scan that is associated with `connection`, and results of the query are stored in `scan`. After evaluation of the first statement by AMOS II, the embedded AMOS II is registered as a client database called “Embedded_DB” to the database server “MBS_DB” in the multi-database federation. The second one ships the “select...from” statement from `MyApplication` to “MBS_DB” through “Embedded_DB”. Thus the MBS data stored in the remote AMOS II server can be accessed from external application programs through the embedded AMOS II.

### 3.3 Visualization of the MBS Orientation Information

The MBS orientation data are stored in the `scan` as the results of the above queries. The following code illustrates some of the key steps to visualize the above numerical data with solid models motions in I-DEAS:

```c
double t;

a_getRow(scan, row, FALSE);
t = a_getDoubleElem(row, i, FALSE);

......

errorCode = instanceSequence[ j ] -> ApplyTransform( trans );
```

In the above code, `row` is a tuple handle, `a_getRow` and `a_getDoubleElem` are functions of AMOS II C interface to respectively obtain rows in the `scan` and the `i`th argument in the `row` as a double precision floating-point number, `trans` is a 4*3 array whose values are assigned by `t`, `ApplyTransform` is a method of Open I-DEAS class `instanceSequence` to apply the input transform to the instance. Therefore the MBS orientation data presented in MECHAMOS in numerical form can be visualized with bodies’ motions in I-DEAS.

The model from the earlier work, i.e. the lifting system of a wheel loader shown in Figure 5, is also used as the multibody system example in the current work. The AMOS II* is populated with orientation data that is previously generated by MECHAMOS using the same lifting system model.
3.4 An Example of a MBS Analysis Data Schema

The I-DEAS model of the lifting system contains necessary information for populating the AMOS II database with data such as inertia properties, mass, and volume of the bodies as well as orientation information governing the motion of the bodies. The current database schema for this example is built upon the existing Open I-DEAS objects representation and is defined using AMOS II types, functions, and objects. There are a total of 15 different types defined in the schema that are associated by approximately 100 stored, derived, and foreign functions. The
The extended entity-relationship diagram of the schema is shown in the appendix. Some selected types and functions are described below:

- **Instance** – defines a supertype for PartInstance and AssemblyInstance. Its attributes include ID, name, and level that correspond to label, name, and hierarchy number of instances in an I-DEAS assembly instance.

- **Transform** – represents a 4*3 transformation matrix that is used to hold the orientation information of an instance. Two functions, RelativeTransform and CumulativeTransform, connect this type with the type Instance. When they are invoked, two transform objects that respectively represent relative transform and cumulative transform of the instance are returned.

- **Property** – contains a set of functions to evaluate the type Part, e.g. volume, mass, openSurfaceArea, and solidSurfaceArea. By calling certain functions, objects representing the centre of gravity and inertia of a part can be accessed. This type connects the type Part with the function physicalProperties.

Furthermore, some other data types, such as 3D_model, assembly, configuration, material, boundingBox, inertia and units are also defined in the database.

This data schema takes advantage of the object-oriented data model provided by AMOS II and uses the type hierarchy. Therefore, the subtype can inherit all functions defined on its supertype. A graphical view of the MBS analysis database schema and some of its instances are shown in Figure 7. This is viewed using the multi-database browsing capabilities in GOOVI from a remote AMOS II server. GOOVI can also be used to query and update this database and the schema. Type Part and Assembly, subtypes of the type 3D_model, inherit all four functions defined on the type 3D_model, i.e. Version, Date, and Name. Types are connected by a certain set of relevant functions. The type Part has the function physicalProperties that returns a type Property. The Property can use the function gravityCenter to obtain the gravity centre of a part that is an object of type 3D_point. Hereby, by using object-orientation, MBS analysis data can be naturally and entirely represented and stored in the database without any loss of information.

### 3.5 Populate and Query I-DEAS MBS Analysis Data

Based on the similar way as shown in section 3.2, I-DEAS MBS data can be populated in the embedded AMOS II.

```
sprintf(str,   
    "create Part(name, Version) instances :%s ('%s','%d');",   
    'p1', 'part1', 2);   
a_execute(connection, scan, str, FALSE);  
```
In the above code, \texttt{str} is a string that is passed to AMOS II for evaluation. After evaluation of the \texttt{create} statement by AMOS II, a part object \texttt{:p1} is created; its name is \texttt{part1} and its version is 2.

![A graphic view, using GOOVI, of the MBS analysis database schema in AMOS II](image)

By doing this, the engineering application gains access to general database capabilities such as storage management, uniform data model, meta data, query language, query processing, and transactions. Users can therefore take advantage of object-oriented query language to compose queries for retrieving, combining, filtering, and transforming data with very high flexibility. Some examples are shown in this section.

Object identifier (OID) is a unique system-generated identifier used to refer to a specific object in the database. The following query gets the OID, Name and the latest Version (version number) of each \texttt{Part} that are used in the multibody system. This is the simplest query in the work since the name and version of the part are the stored functions directly associated with a part object.

```
select p, name(p), Version (p) from part p;
<# [OID 819], "Bellcrank", 1>
<# [OID 837], "Cylinder_2", 1>
<# [OID 849], "Piston_1", 1>
```
Selecting several items and using related functions could combine associated information that belongs to different analysis domains. In the following query, \texttt{partInstance} indicates an instance of the part used as a component in the assembly, and \texttt{level} indicates the hierarchy number of the \texttt{partInstance}, i.e. which level of the assembly it is located at. This query gets the OID, name, hierarchy number, creation date, and material of the \texttt{part instance} that is associated to the \texttt{part}.

\begin{verbatim}
select pi, name(pi), level(pi), Date(p), name(m)
   from partInstance pi, part p, material m
   where owingPart(pi) = p and madeOf(p) = m;
\end{verbatim}

Orientation information governs the spatial positions of each instance in an assembly. It is represented in I-DEAS by a transform that is a 4*3 transformation matrix consisting of a 3*3 rotation matrix and a 3*1 translation matrix. One can extract the orientation information of a part instance for a specific \texttt{part} in different configurations, therefore to evaluate instance trajectories, positions and analyse its motion trend. Based on the highly flexible and extensible object-oriented query language, AMOSQL, users can freely define AMOS II \textit{derived functions} in terms of other functions to accomplish the above complex requirement. The advantage of using derived functions is that by taking advantage of pre-defined functions, queries performing complex tasks can be expressed and interpreted in a convenient way.

create function transforms(partInstance pi) -> transform as
   select trans
   from transform trans, partInstance pi
   where cumulativeTransform(pi) = trans;

create function position(part p, configuration c) -> transform t as
   select t
   from part p, partInstance pi, transform t, configuration c
   where owingPart(pi)=p and
   transforms(pi)=t and
   inConfiguration(t)=c;

position(:Bucket, :config1);

The above query selects the OID that refer to the object of the type \texttt{transform} containing the name, all matrix values of the transform. This transform belongs to the \texttt{part instance} whose part is “Bucket” and in the configuration “config1”.

Furthermore, AMOS II aggregation functions, e.g. \texttt{count}, \texttt{maxagg}, and \texttt{minagg}, can be used to compute aggregate values over sets. The following queries calculates the total mass [kg] of the whole multibody system,
create function MASS(part p) -> real as
    select mass(pr)
    from property pr, part p
    where physicalproperties(p) = pr;

sum(select MASS(p)
    from part p, partInstance pi
    where owingPart(pi) = p);
11930.2

Adding certain constraints can filter the query results. This is provided by AMOS II system functions, e.g. \texttt{min}, \texttt{max}, +, -, *, / . The query below selects the parts, their instances for the level with hierarchy number 2, and the mass of their corresponding parts that is greater than 2000 kg.

\begin{verbatim}
select p, name(pi), MASS(p)
from part p, partInstance pi
where owingPart(pi) = p and
    MASS(p) > 2000 and
    level(pi) = 2;
\end{verbatim}

<# [OID 843],"Lifting_arm_21",2407.31>
<# [OID 843],"Lifting_arm_22",2407.31>
<# [OID 825],"Bucket_40",4087.84>

It is not actually necessary to store all types of MBS analysis data in the database since AMOS II derived or foreign functions can be defined to compute them. Foreign function is one of the most important features to extend AMOS II functionality. It can be implemented in an external programming language such as C, Java, or LISP. A simple example will here illustrate how to define a foreign function in AMOS II using C as the implementation language.

In the C code, a simple foreign function calculating part density is defined as,

\begin{verbatim}
void density_c(a_callcontext cxt, a_tuple t)
{
    double v, m, d;
    m = a_getdoubleelem(t, 0, FALSE);
    v = a_getdoubleelem(t, 1, FALSE);
    d = m/v;
    a_setdoubleelem(t, 2, d, FALSE);
    a_emit(cxt, t, FALSE);
    return;
}
\end{verbatim}

The following are the AMOSQL statements to associate this foreign function with AMOS II function,
create function Density(part p) -> real d as
    select calculate_density(MASS(p),volume(p));
Density (:Bucket);
7820.0

The above query takes part as input and returns density of the given part. In this query, AMOS II foreign function calculate_density is one of the C functions in the application program. It binds with the derived function Density and takes mass and volume of the part as inputs to calculate density. When the function Density is called, calculate_density will be invoked through a mechanism provided by the AMOS II C interface. After calculating, it returns the result – the value of part density.

4 Conclusions and Future Work

Product development requires product information to be communicated and integrated between computer-based analysis tools in a globally distributed and heterogeneous computing environment. Generally, engineering information involve a higher level of complexity than information that is found in traditional database system for administrative applications. The product development process normally involves a large set of different software programs that run on different platforms. This raises a demand to be able to access product data in a system- and application-independent manner. Thus, it should be possible to integrate product data in such a distributed and heterogeneous EIS environment.

This paper shows how modern database technology can be used to support the integration of engineering information, specifically concerning multibody system analysis information. The example provided shows how to use an object-relational database management system as an embedded DBMS to efficiently integrate and manage MBS analysis information. It also shows how to obtain MBS analysis information from an engineering data source in a distributed environment using a CORBA-based client-server model.

The benefits of this work include:

1. Integrate engineering information between different analysis domains. Although we only focus on the data integration on multibody system analysis domain, in general this approach is also applicable to integrate engineering information between other analysis domains such as FEA, CFD.
2. The integration of the systems is accomplished at the query language level. This means that the applications are closely integrated and no file transfers are needed.
3. Engineering data can be stored entirely without losing information since the extensible object-oriented data model naturally fits with the object-oriented data structure as found in engineering data.
4. All the data is immediately accessible through the query language, meaning that users have high flexibility to retrieve, combine, filter, and transform the data that
originates from different representations and data sources from different domains. This is much harder to achieve in, for example, the MCAE system itself.

Currently solid models and mathematical models have to be built independently in I-DEAS and MECHAMOS, i.e. there is no connection concerning the geometrical data of I-DEAS solid models and mathematical models in MECHAMOS. It is user who maps them together. Furthermore the MBS analysis data queried by I-DEAS is pre-generated by MECHAMOS and stored in one AMOS II database. The future work should enable the two systems to be integrated seamlessly, i.e. MECHAMOS uses geometrical data of I-DEAS solid models to build mathematical models and I-DEAS sends queries directly to MECHAMOS to retrieve information concerning the MBS behaviour and visualizes the MBS motions.

STEP (STandard for the Exchange of Product model data) is a well-known ISO standard aiming at neutrally and system-independently describing products throughout their life cycle. A large benefit can be gained by introducing STEP standard to DBMS area as STEP provides widely-accepted and mature data representation for engineering data to be accessed, combined and exchanged from applications. However considerations on balancing data representation and data processing between STEP and database technology should be further addressed in order to both represent and process data efficiently. The further development of this work should be adding the STEP layer between the two systems. Therefore the MBS data generated by MECHAMOS and I-DEAS could be investigated as standardized plug-in in a integrated EIS.

5 Acknowledgements

This research work was financially supported by the Swedish Engineering Design Research and Education Agenda (ENDREA) and the Polhem Laboratory at Luleå University of Technology.

References


Appendix:
Extended Entity-Relationship diagram of the MBS analysis database schema in AMOS II
Abstract

Today, work collaboration is normal practice in developing modern products. Engineering collaborative work involves a number of team members that need to share and exchange design ideas while working with engineering analysis tools such as mechanical computer aided engineering systems. This work presents the M-Sync prototype system that uses an active database approach to enable exchange of engineering information among distributed team members in a timely manner. The distributed data is fully accessible by the local member and is automatically synchronised between different places using a database management system that support event-condition-action (ECA) database rules. Only updates introduced at one location are distributed to other locations, thereby minimizing information transfer and enhancing performance. Members working at different locations can therefore work in a peer-to-peer (P2P) manner and interactively manipulate the same set of information at the same time.

Keywords: engineering information, engineering collaborative work, active database management system, ECA database rule, synchronisation, distribution, peer-to-peer

1 Introduction

Modern products are often the outcome of a collaborative development effort of many developers originating from many different engineering domains. These development teams are often scattered among large enterprises. The development of computer based tools and networks to support development processes has improved the potential to exchange information between teams situated in different places.

Product development projects are often organised jointly with the subcontractors from large enterprises. Most joint development projects have the computer-aided-
design (CAD) model as a common base during product development. These CAD models are then used in other domains in the development process. The structural analysis is performed with finite element methods (FEM) while the suspension is analysed by multi body system (MBS) analysis. Furthermore, body development can perform computational analysis to predict the airflow by using computational fluid dynamics (CFD) tools, such as optimising the placement of the air-conditioning system. All parts of these development processes interact in the development of an excavator as outlined in Figure 1. This type of distributed work requires the existence of computer-based tools that enable the sharing of information through computer networks.

**Figure 1.** Collaborative work involving several engineering domains

Furthermore, these engineering activities are often dependent on one another. A minor modification introduced by a team member in one of these domains can affect the parallel and down-stream work of other members in other domains. For example, updates made to a CAD solid model can affect both subsequent CFD and MBS analyses. It is therefore important that designers can collaborate during the development, such as exchanging ideas and reviewing different design suggestions when they work on the same product. It is also important to have the capability to efficiently share and exchange engineering information among team members to support this type of distributed collaborative work environment.

One alternative to support distributed collaborative work is to use programs supporting “shared applications” such as NetOP, Netmeeting, and MSN Messenger.
These programs can share an application by distributing its screen images. Another alternative approach let users share product information in terms of a VRML-based (virtual reality modelling language) geometric model [1, 2]. Some of these systems also offer capabilities to update information remotely. This type of information sharing is achieved on a “view” level meaning that the information exists only at the host location and that clients can only visualize the information locally. If the clients need the information it has to be exchanged separately outside the program. Consequently, the limitations of this type of passive-interactive programs are that only one person at a time can update the information and that the engineer at the host controls the update access.

The “shared applications” approach can, to some extent, meet the requirements for collaborative design. However, to improve interaction between team members, it is desirable that the principal design tools, such as CAD systems, be also integrated and that information sharing is implemented on a “data” level. One way to exchange information is to distribute the whole model file to all team members. This can be accomplished either via a native format between the same CAD systems or via some standardized format such as STEP or IGES, between different CAD systems. A problem with this approach is that it can be time consuming to transfer large CAD models. Qiang et al [3] proposed a system to support CAD-based collaborative design through the web. This system, called WPDSS, exchanges information by distributing CAD macro files among team members. The macro files consist of a record of the series of user interactions between keyboard and mouse. This system reduces the data volume to be distributed since the size of macro file is much smaller than a CAD model. However, since the macro files record ALL user interaction with the input devices, they certainly contain information that is unnecessary to be sent to other designers. For instance, a designer creates something unsatisfactory, which he or she immediately deletes and then creates a new one. Such failed operations on the a CAD model will be re-played at other members’ systems. Besides, such an approach would only admit information sharing among CAD systems of from the same vendor where the macro file can be correctly interpreted. Other systems like WISPER [4] and QuickMould [5] enable project information to be shared among team members through a STEP file and a master model respectively. However such file-based information sharing has its inherent drawback because a file is basically for archiving purposes. Normally, it has no, or at least, insufficient functionality for online CAD interoperability regarding access to the CAD data structure and high-level functionality. Consequently, the problem of file-based information sharing is to determine when and what to send to the team members. Does the receiver need to have a new version of the model with the latest updates or is the model that he or she has sufficient? Merging updated models from different development teams can also pose problems.

This work presents a generic approach to support distributed collaborative work by using active database technology as a basis for enabling geographically distributed team members to exchange engineering information in a timely manner. In this approach the distributed information is fully accessible by the local member
and is automatically synchronised between different locations. Updated data at one location is identified locally and then distributed to the other locations by mechanisms of active database technology. Designers working at different locations can, by relying on a distributed peer-to-peer architecture, interactively manipulate the same set of information at the same time. Active database rules, i.e. event-condition-action (ECA) rules, and mediation functionality are used to identify and distribute these updates.

2 Technologies for Distributed Engineering Information Systems

Virtual prototypes have been introduced to act as a common information source in product design. This approach requires an efficient information management that can supply a correct and consistent view of data for each engineering activity while avoiding unnecessary replication of data. For distributed development teams, which are quite common today, this must be accomplished in an engineering information system environment involving a network of distributed and, possibly, heterogeneous sets of computing resources. Kao and Lin [6] have proposed a model of collaborative design where participants in different places can share and modify the same CAD drawing simultaneously. Designers work on a copy of a common data model that is stored in a server acting as a shared database. Zhou and Lin [7] presented a system that enables two CAD users, working at different locations, to work together on a 3D CAD geometry co-editing task via a shared database. A data interchange format and relevant mechanism were developed to identify changes made on the geometry. However, the server, the shared database and the change identification mechanism of the above systems were implemented with a conventional programming language. In comparison with employing a database management system (DBMS) to implement the above features, a conventional programming language works on a lower level in terms of reuse of system functionality, system complexity and maintenance.

By incorporating a database management system to handle the distribution and management of data in an engineering information system (EIS), one directly accomplishes high-level reuse of system functionality that otherwise requires reimplementation since the DBMS can automatically support storage management, meta-data, query processing, and transactions. Furthermore, it can support uniform data modelling and manipulation through the query language. Together, these capabilities can greatly improve the development, use, and maintenance of EISs. Data redundancy can be avoided, data consistency can be controlled, and data manipulation can be optimized and secured using built-in mechanisms of the query language and other subsystems of the DBMS. Some of the above advantages of using database management system to support distributed engineering collaborative work have been embodied in several recent research works [8-10].
In addition, most DBMSs support a variety of interfaces for different host languages and systems and some DBMSs support an active behaviour through rules or triggers. These capabilities make it possible to generalize the conventional view of exchanging data among applications in an EIS to mediation [11] of data between applications and data sources, i.e. multi-directionally sharing data that might involve non-trivial data translations and transformations. A DBMS working as a mediator provides methods to monitor, transform, combine, locate and query data. Again, the built-in features of a DBMS can be reused to distribute and actively monitor engineering information.

2.1 Object Database Technology

Modern database technology is expected to play an important role in engineering information systems in the future [12]. However, the conventional relational database management systems (RDBMS) use a relational data model to store data in the database. In many cases relational DBMS has limitations to efficiently manage data of engineering applications and scientific computing, as they usually require more complicated data structures, representations and operations. Object-oriented database management systems (OODBMS) were proposed to meet the needs of these more complex applications. OODBMS support richer and more extensible data models, such as object-oriented data model. First generation OODBMSs are seamlessly integrated with their corresponding object-oriented programming language and provide persistent storage of programming objects, but it is limited and rigid in terms of built-in hierarchical paths in the data [13] and do not include several RDBMS features such as a declarative query language, meta data management, and views [14].

The current research trend in database community focuses on the second generation OODBMS, called object-relational database management systems (ORDBMS) [12]. ORDBMS combines many features of the object data model and languages into the relational data model. Therefore, it contains features from both RDBMS and OODBMS, such as object identifier, query language supporting object-oriented data models and triggers. Companies in the RDBMS market have released products in this area including, IBM and Oracle. Many research prototypes have also been built, with some of the most well known are HP-Iris, Berkeley-Postgres, and IBM-Starburst.

2.2 Active Database Technology

Traditional database management systems are passive, i.e. data is created, retrieved, modified, and deleted only in response to operations issued by users or application programs. Active database management systems (ADBMS) are event-driven DBMSs. They enhance the functionality of traditional DBMSs so that certain operations can automatically be performed in response to the occurrence of certain events or when certain conditions are being satisfied [15]. Active database technology uses active rules, so called event-condition-action (ECA) rules, as an
integral feature to monitor events that are specified in the rules declaration. When
the event occurs and the conditions are satisfied, the corresponding actions are
executed. Hence, active database systems can recognise specific situations and react
to them without direct and explicit user or application requests [16].

ECA rules are often declared as conditional expressions, like if-then-else or case
statements in conventional programming languages. However such statements are
static meaning that they cannot be changed unless the code is recompiled. The EAC
rules can be dynamically added and changed in databases that support incremental
recompilation of rules and functions. The following simple example illustrates ECA
rule usage. This rule updates the material of a given part:

create rule updateMaterial( part p ) as
   from material m
   on updated( state( m ) )
   when m = madeOf( p ) and updateState( m ) = true
      do updatePartMaterial( p );
activate rule no_high(:cylinder);

Using ADBMS to support different engineering applications to monitor and
control engineering models in databases has been discussed and investigated. Sköld
specified the requirements that computer integrated manufacturing and
telecommunications networks demand on the database technology and presented
some scenarios using ADBMS within these two fields [17, 18]. Andler et al [19]
introduced a method for distributing functions in a real-time control system using
ADBMS. Roller et al developed an active database approach to increase design
process productivity by informing designers about the correlations to parallel work
and by practising collaborative work [20]. Other works [21-24] further show the
usage of ADBMS within different fields such as process industry, software
integration and workflow management.

2.3 Common Object Request Broker Architecture

Common object request broker architecture (CORBA) [25] is an architecture that
enables distributed program objects to communicate between one another regardless
of what programming language they were written in or what operating system they
are running on. CORBA communicates objects via its central component, the object
request broker (ORB), which provides broker services between clients and servers,
such as target object location, connection management, message delivery, and
method binding. Clients send requests to the ORB asking for services provided by a
certain server (regardless of platforms) and the ORB locates the server, passes the
message, receives the result, and delivers it back to the clients. The functionality of
an object is described using the interface definition language (IDL) that is a host-
language dependent and declarative language. The IDL language has mapping
standards to major programming languages such as C, C++, Java and LISP. CORBA
therefore provides programmers with a transparent way to communicate objects in a
heterogeneous and distributed environment.
2.4 The Peer-to-Peer Computing Model

Peer-to-peer (P2P) is a distributed computing model in a decentralised architecture. Every entity in the network, referred to as a peer, can either request or provide a service. This means that a P2P network distributes information among the peers instead of concentrating it at a single server, as in a client/server computing model. This paradigm offers exciting advantages in information sharing [26, 27] and could be a suitable computing model in a collaborative work environment. For instance, geographically distributed teams of engineers, designers, or scientists could directly share and exchange technical documents, drawings, and models. They could broadcast updates to working documents to all team members and discuss them with instant messaging services [28].

2.5 The AMOS II Database Management System

The AMOS II (Active Mediator Object System) DBMS is used in this work. AMOS II is a main-memory resident and object-relational database management system [29]. The AMOS II kernel is an object-oriented and extensible DBMS kernel that includes communication primitives that allow a set of AMOS II servers to be configured into a distributed P2P architecture [30, 31], which can handle complex data structures such as engineering data. AMOSQL is an object-oriented query language for AMOS II used for defining, populating, querying, and updating the database. AMOSQL is similar to the object parts of SQL-99 [32] and is based on the functional query languages OSQL [33] and DAPLEX [34], with extensions of mediation primitives, multi-directional foreign functions, late binding, active rules, etc [35]. Furthermore, AMOS II is a lightweight main-memory resident DBMS and it is possible to embed it into, or interface it with, other systems such as a MCAE system, using its interfaces for C, Java, or LISP. The functionality of a system can therefore be extended with DBMS facilities such as storage management, uniform data model, query language, and query processing.

The AMOS II data model consists of three basic concepts - objects, types, and functions - corresponding to instances, classes and methods in an object-oriented programming language. Basic functions can be classified into stored, derived, foreign, proxy functions and database procedures. Stored functions represent properties of objects stored in the database and proxy functions represent functions in other databases. The derived functions are functions defined in terms of object-oriented queries over other AMOSQL functions while database procedures are defined using a procedural sublanguage of AMOSQL. Foreign functions are implemented through an external language such as C, Java or LISP.

AMOS II is an active DBMS. The rules are first-class objects and of the type “rule”. AMOS II supports ECA, event-action (EA), and condition-action (CA) rule models. The AMOS II rule processor handles rule creation, deletion, activation, deactivation, tracing, and execution with the corresponding AMOSQL commands: create rule, delete rule, activate rule, deactivate rule, trace_rules and ruleCheck.
The events that can be specified in AMOS II are updated, added, removed, created and deleted. The updated, added and removed event types monitor updates to stored and derived functions. The created and deleted event types monitor the creation and deletion of an object instance. Events can also be composite.

Furthermore, AMOS II is a P2P mediating system where AMOS II mediator servers communicate over the Internet using a TCP/IP socket-based protocol. AMOS II mediator servers have DBMS facilities, such as a local database, a data dictionary, a query processor, transaction processing, and remote access to databases. One limitation is that a mediator server currently cannot be reverted to a mediator client. AMOS II uses a wrapper approach to interface data sources with other representation formats, i.e. the wrapper has knowledge of other data sources, such as relational data sources, XML and STEP/EXPRESS. The features of AMOS II make it possible to access, combine, and manipulate data from different and distributed data sources through AMOS II peer mediators.

3 The M-Sync Prototype System

3.1 The M-Sync System Overview

A prototype system, M-Sync, has been constructed to study distribution and synchronisation of engineering information. Its architecture is outlined in Figure 2.

![Figure 2. The M-Sync system architecture](image-url)

The system consists of one AMOS II name server and peers that are distributed at different locations. AMOS II mediators are interfaced with a CAD system at each
The mediator is used to identify updates, notify other peers, and receive updates made by other peers. The mediators use the *name server* to keep track of the peers and the network to communicate between them. The AMOS II mediators can then communicate directly to one another since the *name server* provides the names, locations, and other general data regarding the mediators. The configuration is shown in Figure 3. When the design process starts, M-Sync automatically registers the local peer mediator into the multi-mediator federation. A designer issues a “sync” command to the M-Sync system after completing a CAD model modification. The *data loader* downloads the CAD model into the mediator. During this procedure, ECA database rules are activated to monitor the data and detect updates. Once a change is detected, the ECA rules trigger database functions to modify the state of the updated CAD entities for use in change identification and peer notification later on. The state can be “created”, “deleted” or “modified” according to the fact. After downloading the whole model, the *update identifier*, which contains a set of ECA rules and database functions, automatically identifies the updates on the CAD model, and then triggers the *peer detector* and the *peer notifier* to detect the registered peers and propagate the updates to the remote mediators. On the other hand, when a mediator receives updates made by other designers at other locations, the *MCAE synchroniser* that also contains a set of ECA rules and functions uploads the data to the MCAE system.

This approach enables developers to simultaneously work on the same model even if they are situated in different places. The information can automatically be synchronised between different places by identifying updates at one location and transferring them to the other locations where they are synchronised with other MCAE systems. This makes it possible for multiple users to interactively manipulate the same set of information. The design tool presently used in the system is the MCAE system I-DEAS from EDS. The Open I-DEAS application program interface (API) is used to access and download I-DEAS model data to AMOS II. Open I-DEAS is a CORBA-based API that enables users to build client-server applications to directly access and manipulate I-DEAS data. In this work both I-DEAS and AMOS II run under Windows XP.

Compared with the systems presented in [6, 7, 9] that use a shared database to store and manage the CAD model, this approach enables design team members to work in a P2P manner and have full access to the product model since all the necessary data are available locally. Designers can work on their own model concurrently. This differs from the shared database approach where a designer has to wait until other members release the editing right in order to change the model. However data concurrency among members could be much harder to control.

If several members issue the sync command at the same time, there is a risk that the system hangs. In the current M-Sync system, a simple locking mechanism is implemented that resolves this transaction problem. The procedure is separated into 5 steps:
1. A designer issues a sync command to the system

2. The system first sets the state of the local mediator to be “busy” in the name server

3. The system queries the name server to find the registered mediators

4. If the registered mediators are “free”, the system commits the synchronisation to them. If the registered mediators are “busy”, the system aborts the synchronisation and keeps those already identified updates in the database. When the states of the mediators are “free”, these updates can be distributed to them.

5. Finally the system resets the local mediator to be “free” so that it can receive updates made by other mediators.

![Diagram of M-Sync system configuration]

**Figure 3.** The configuration of the M-Sync system

### 3.2 An Example of a MBS Assembly

A simple assembly, shown in Figure 4, a lifting system of a wheel loader modelled and assembled in I-DEAS, is investigated to verify the validity of the proposed approach. In the system’s present state all peers must start with the same
set of solid model information. This model consists of a bucket supported by two
identical articulated arms. Four hydraulic pistons govern the movements of the arms
and other executable components. In addition, all the components are connected by
joints.

The objective to be monitored is part orientation information. Orientation
information governs the spatial positions of each part instance in an assembly. It is
represented in I-DEAS by a 4x3 transformation matrix of double precision values
consisting of a 3x3 rotation matrix and a 1x3 translation matrix, illustrated in
Equation 1.

In this example, a database schema is defined that conforms to the I-DEAS native
data structure, i.e. a CORBA IDL-based (interface definition language) description
[36]. If information should be exchanged by different systems, it could be
advantageous to employ industry standards like STEP to define a common database
schema. This is beyond the scope of this paper. By using Open I-DEAS and the
AMOS II C interface, all the 12 values in the matrix can be directly accessed in I-
DEAS and stored in AMOS II for identifying updates later on. Figure 5 shows a
flowchart of the auxiliary working procedures performed when a solid model is
modified.

\[
\begin{bmatrix}
R_{11} & R_{12} & R_{13} \\
R_{21} & R_{22} & R_{23} \\
R_{31} & R_{32} & R_{33} \\
T_x & T_y & T_z
\end{bmatrix}
\]

(1)

**Figure 4.** Solid model of the lifting system
3.2.1 Identifying Data Updates

When an engineer, for instance at peer 1, completes position adjustment on the parts of the lifting system and issues a sync command, the orientation data of all the parts belonging to this assembly are downloaded to the AMOS II mediator where update identification is performed. Stored functions, foreign functions, and ECA rules in the update identifier are created and used to actively identify updates.

As shown below, the AMOS II type transform is created as a subtype of type MCAE_object, whose property includes a Boolean number need_update and a string name. The type transform therefore inherits its properties, i.e. need_update and name. Furthermore, the type transform has a stored function called value, which contains a vector of real numbers corresponding to Equation 1.

\[
\text{create type MCAE_object properties( need_update boolean, name charstring );}
\]
\[
\text{create type transform under MCAE_object;}
\]
\[
\text{create function value( transform ) -> vector of real as stored;}
\]

The following code creates a function to compare the 12 values in transforms a and b. If any corresponding value is not equal that means that a change has been detected, the old value is replaced by the new one and the attribute need_update of the original transform b is set to be “true”.

\[
\text{create function cmpTrans( transform b, transform a )->boolean as}
\]
\[
\text{for each integer n where n=iota( 0, 11 )}
\]
\[
\text{if value( b )[ n ] != value( a )[ n ] then}
\]
begin
    set value( b ) = value( a );
    set need_update( b ) = true;
end;

The following AMOS II rule is created to identify updates on the transform. By calling the function `cmpTrans`, this rule identifies if the newly created transform differs from the original one.

```
create rule updateTransform( transform b ) as
    from transform a
    on  created( a )
    do
        begin
            cmpTrans( b, a );
            delete a;
        end;
```

In the following code, the type `MCAE_server` represents the MCAE system to which the AMOS II mediator is interfaced. If the attribute `need_update` of `MCAE_server` is `true`, i.e. the engineer made some updates on the CAD model, the rule `check_MCAE_Server` then calls foreign functions `detectPeers` and `notifyMediators` to detect registered mediators and to notify them about the updates.

```
create type MCAE_server under MCAE_object;

create rule check_MCAE_Server( MCAE_server s ) as
    on updated( need_update( s ) )
    when need_update( s ) = true
    do
        detectPeers( );
        notifyMediators( );
    end;
```

### 3.2.2 Detecting Peers and Synchronising Data

When updates are identified, the foreign function `detectPeers` detects registered peers and sends them the transforms that have been updated. The basic idea is straightforward. First, it queries the AMOS II name server about the registered mediators, i.e. which mediators have registered to the name server. If mediators are found and their states are “free”, it then calls `notifyMediators` to collect all the transforms that have been updated and sends them to the registered mediators except the name server and itself.

Furthermore, AMOS II listens to inter-database requests from other AMOS II systems when it is in “listen” mode. The AMOS II mediators are in “listen” mode all the time so that they can immediately respond to a notification. After being notified, the mediator calls foreign functions in the MCAE synchroniser to receive data and then transfer the data to the MCAE system, as shown in Figure 6. Consequently an engineer at any peer can immediately observe updates made by other engineers.
4 Conclusions and Future Work

Team members working in distributed engineering collaborative work need to have the capability to share and exchange design information and ideas. This work presents an active database approach to enable exchange of engineering information among distributed team members in a timely manner. A prototype system, M-Sync, is constructed to support mechanical computer aided engineering (MCAE) where co-located engineers engaged in the same engineering design work can interactively and simultaneously work on the same solid model. By interfacing a database management system that supports active database rules and peer-to-peer communication, information sharing among team members and MCAE systems can be supported. Updates that are made in one MCAE system are actively identified and only updates are distributed to other peer MCAE systems. This means that the information residing in the distributed MCAE systems is automatically synchronised. A simple locking mechanism is also implemented that resolves user-level transaction problems, i.e. avoiding conflicts caused by concurrent synchronisations issued by multi-users.

The benefits of the current approach include the following:

- Active database technology provides mechanisms to monitor, transform, locate, and query data between distributed engineering applications. This results in more powerful methods for sharing engineering information. Together with built-in communication primitives of the DBMS, a distributed P2P engineering information system is supported.

- This approach ensures that all team members have up-to-date and consistent information as database rules actively monitor the data, and database mediators immediately respond to inter-database requests. It can also improve the functionality of distributed engineering applications since every team member can have full access to all information locally. This model reduces the necessary data transfer during updates by sending only updated data to other team members in comparison to approaches where the complete model is distributed at updates.
• General capabilities of the DBMS, such as storage management, data modelling, query language, query processing, transactions, and meta-data can provide easier and more efficient data management compared to equivalent implementations in conventional programming languages.

Future development needs to consider the following issues:

• Performance studies must be carried out. The current approach that relies on main-memory and scalable database technology is expected to provide efficient update identification.

• Transactions are currently used in the DBMS but further studies are required to completely support concurrency control for engineering activities on the data level. This might also include capability for version control that is a quite complex topic in engineering design.

5 Acknowledgement

The authors wish to acknowledge the Polhem Laboratory at Luleå University of Technology, which is financed by VINNOVA, partner companies, and Luleå University of Technology. We also acknowledge the Swedish Engineering Design Research and Education Agenda (ENDREA) financed by the Swedish Foundation for Strategic Research.

References


[29] http://www.dis.uu.se/~udbl/


Paper D
M-Sync: A system for distributed collaborative design

Haoxue Ma

Abstract

In recent years, the increasing trend towards manufacturing globalisation has led to the need for collaborative CAD/CAM systems. However, traditional CAD/CAM systems are stand-alone and single-user applications where users typically interact with local systems that do not properly meet the requirements of distributed multi-user collaborative applications. This paper presents an M-Sync system that employs an object-relational database management system (DBMS) supporting active databases and a peer database architecture to extend functionalities of the current CAD system. It is shown that database technology not only provides direct support for the effective storage of and efficient access to complex CAD data, but also assists collaborative design activities. Specifically, the DBMS is used to identify and propagate data updates made by the CAD system and maintain data consistency between the CAD system and the database. This work also employs the distributed object computing to access, operate, and distribute CAD system objects between the CAD system and the DBMS in a platform independent manner. The system architecture and basic transaction management are also studied. The present work facilitates synchronous collaborative design where distributed designers perform online interactive discussions regarding incremental modifications of a CAD system model. The proposed approach is implemented as a collaborative CAD system environment where distributed design information can be actively synchronised between co-located designers when typical collaborative design tasks such as solid and assembly modelling are being performed.

Keywords:
distributed collaborative design, update identification and propagation, consistency control, object-relational databases, active databases, peer databases, CAD systems

1. Introduction

In recent years, the rapid development of computer aided design (CAD) and computer aided manufacturing (CAM) has dramatically changed the product development process. CAD/CAM system enables engineers to realize product concepts in the early stage of product development without physical parts actually being manufactured. Using such advanced tools greatly improves the product quality and shortens the development lead-time. The increasing trend towards manufacturing globalisation has led to the need for collaborative CAD/CAM systems [1]. In a collaborative CAD/CAM system, several co-located designers
work in the same or different domains are normally involved in collaborative work on a common design activity. Information communication between both distributed users and CAD/CAM systems has become one of the most critical factors in ensuring successful collaboration. However, traditional CAD/CAM systems are stand-alone single-user applications where users typically interact with local systems that do not properly meet the requirements of distributed multi-user collaborative applications.

As a transitional approach, groupware [2] such as e-mail, shared whiteboard, shared applications, video conferencing, and 3D CAD model viewer have been extensively used to support current collaborative design work. To some extent, groupware meets collaborative work requirements since it provides functionalities that allow users to exchange and share product information by means of either product model file or multi-media in a synchronous or asynchronous manner. Therefore, “what you see is what I see” can be achieved between co-located designers. Some of them even offer capabilities to remotely update product information one user at a time. Typically, procedures using groupware to communicate occur as follows: when local information is requested by remote designers, a local designer must first save the design work before sending a copy of the current work or standardized format such as STEP, by e-mail or FTP, to the remote designers; or by using a shared application or whiteboard to share the current work with other designers. Obviously, interactivity of this working manner is limited. Furthermore, CAD system models contain large amounts of high-level design information. Performing real-time online discussion and modification on such information are essential for designers; this is especially apparent when designers perform closely-coupled collaborative design tasks. However, information sharing supported by groupware is on either a static or “view” level, since groupware has no, or at least insufficient, functionalities to support collaborative product design in a concurrent and interactive manner, as discussed in [1, 3].

Information systems that support CAD/CAM collaborative design have been studied and discussed in recent years. CAIRO [4] is a multimedia conference system for distributed collaborative work, focusing mostly on solving social issues such as speech request, speech duration, and chairman control. Technical issues dealing with engineering data are not specified. COCADCAM [5] is a collaborative environment supporting NURB surface co-modelling and tool path generation for remote machining, but lacks the functionality for solid modelling. The Virtual Reality Modelling Language (VRML) is the 3D equivalent of HTML for sharing and displaying 3D objects on the web. Several systems use VRML to share 3D objects in collaborative work. TeCo3D [6] shares a VRML-defined 3D model between collaborative designers using a VRML browser. CyberEye [7] uses a Java 3D-based 3D modelling browser, CyberEye Viewer, to share VRML files that are stored on a server. As exemplified, VRML can be used to provide a shared view of 3D objects in an onscreen virtual-reality environment. However, VRML was not intended as a means for exchanging CAD data since several important design-modelling features are missing such as dimension and tolerance modelling. Hence, it
has not been employed by CAD systems for design modelling purposes [8]. Furthermore, VRML does not support updates that are important in collaborative design. That means that if a CAD system should use VRML for sharing model data, a new VRML file must be generated each time that the CAD model is updated and the old file should be replaced by the new one.

The Extensible markup language (XML) is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications. Su et al. [9] proposed an open collaborative CAD system model based on Java/CORBA/XML. XML is used to describe features of and relations between primitives of simple solid model and for design of a language for defining feature design history. In this system, sending XML-based semantic messages supports distributed design activities, thereby decreasing the necessary data volume transferred via the network. However, when representing complicated objects, especially CAD models, the strict XML specification inevitably leads to very large XML documents. The direct drawbacks are the heavy overheads from both network transfers and the parsing procedures. Furthermore, XML’s node-based query and update procedures for XML data also appear to be insufficient for such large documents. Lu et al. [10] proposed a web-based 3D animation environment for distributed users. However users only get animations compressed by using a sequence of static mpeg images. Online interactive data exchange and discussion are not supported.

When performing closely-coupled collaborative design work, co-located designers need to frequently exchange design information with each other. It is certainly desirable to propagate only the updated data to others designers rather than sending the whole CAD system model. However, it is typical that not all functionalities of a CAD system are provided through its Application Programming Interface (API) for external users to access. For instance, a CAD system normally has no change detection mechanism implemented on an API level.

This paper presents an M-Sync system that uses an object-relational active database management system (DBMS) and a CAD system to conduct synchronous distributed collaborative design. The functionalities of a stand-alone and single-user CAD system are extended by interfacing the DBMS. The DBMS is not only used to store, but also to query and manipulate engineering information, for instance, identify and synchronise updates made by the CAD system. Furthermore, the DBMS also manages consistency control between the DBMS and the CAD system, as well as basic transaction management. More specifically,

- Database active rules are used for identifying and propagating updates made to engineering data. The rules are also used for data consistency control and simple transaction management.
Multi-database primitives are used to design a system with a peer-to-peer (P2P) architecture where CAD systems can directly communicate among one another via their interfaced databases.

An extensible object-oriented query language is used to define the database schema, store, populate, and manipulate engineering data.

Furthermore, distributed objects are accessed, operated, and distributed in an operating system-independent manner by employing the Common Object Request Broker Architecture (CORBA). In this system, distributed information is fully accessible by the local designer and is automatically synchronised between co-located designers. Updated data at one location is identified locally and then distributed to other locations. This system supports the distributed collaborative design in a synchronous and incremental manner. Co-located designers can interactively and simultaneously manipulate the same set of information.

2. An overview of the system

This section describes the M-Sync system architecture design, enabling technologies, and system configuration, as well as some technical issues for constructing the system.

2.1 The system architecture

It is well known that stand-alone, single-user CAD systems have many limitations regarding engineering collaborative design, which explains the popularity of developing information systems aimed at extending the current CAD systems to be distributed multi-user applications. There are basically two kinds of system architectures for such collaborative systems, viz. a centralised model, referred to as the client/server architecture where all shared data is maintained at a single location, and a replicated model, referred to as the peer-to-peer architecture where each site maintains and processes a copy of shared data.

2.1.1 The centralised model

The centralised model employs a central database, i.e. a server, to manage engineering information. Co-located designers download design information, for instance a CAD system model, from the server and edit the model using local CAD programs. Figure 1 shows the work procedure for this type of collaborative work.
If a certain designer wants to edit a CAD model, they must first copy the model in the local system (check-out), and then start the editing task. During editing, the model is locked by the database system and can only be modified by the current designer. If the designer has made modifications on the model, they must update the corresponding model stored in the central database (check-in), unlock the model, and send messages to the other designers about this update. Other designers must therefore re-download the modified model to proceed to the design task. The key advantage of the centralised model is that it is easier to ensure data consistency as there is only one master copy saved in the central database. However, a problem is that designers have either only shared view or individual access to CAD system models [11], meaning that one designer has to wait until other members release the editing right to change the model in the server. Performing collaborative work using this model typically requires high network bandwidth, since the whole model has to be re-downloaded by other designers even if there is just a minor update, such as one dimension of a solid model, made on the master model. Besides, it is potentially less fault-tolerant than replicated model because the central host is a single point of possible system-wide failure [12].

2.1.2 The replicated model

The replicated model is adopted in this work and does not have a central database that contains master models. Instead, each node contains a copy of the shared data in the local system so that designers have full access to the data at any time. Each node also has equivalent capabilities as well as responsibilities, i.e. either request or provide service, thereby differing from the centralized model where the central database is dedicated to serving the others. Modifications made on the local model are identified and then propagated to other collaborative designers in the form of native or standardized CAD system data, as shown in [13-15]. The data volume to be distributed is reduced since only the modifications are transferred rather than the whole modified model. The different sites could be synchronised by applying, interpreting, or executing the data or design message sent by the modifier. Additionally, designers can perform independent simultaneous work and have different views of the shared data since the data resides locally; therefore, designers
can freely modify the model and merge the changes with others afterwards. However, maintaining data consistency among distributed replicas is more complex than with the centralized model.

2.2 Enabling technologies for the system

2.2.1 Database technology

Much research [16-19] has studied coupling a database with a traditional CAD system to extend the CAD system functionality. It has been shown that general database features can satisfy requirements of advanced engineering applications.

- Object-oriented data model. Engineering applications need efficient representations of objects having complex structures, operations, and methods. An object-oriented data model naturally fits with the object-oriented data representation and is a rich and extensible model that can efficiently represent and manipulate such complex data.

- Active database rules. Monitoring and controlling events, and then performing the relevant actions are critical for many engineering domains such as computer-integrated design and manufacturing, telecommunication network, and real-time control systems. An active database uses active rules, so called event-condition-action (ECA) rules, as an integral feature to monitor events that are specified in the rules declaration. When the event occurs and the conditions are satisfied, corresponding actions are automatically executed. Hence, active database systems can recognise specific situations and react to them without direct and explicit user or application requests [20].

- Management on distributed data. The distributed database system emerged as a merger of two technologies: (1) database technology and (2) network and data communication technology. It provides capabilities for local autonomy, transparency, reliability, and easier expansion, i.e. very important issues for distributed collaborative engineering applications.

- Wrapper-mediator approach. It integrates heterogeneous data by adding a wrapper-mediator between data sources and applications. CAD data are normally used as input by different down-stream analysis activities, e.g. geometrical data is the basis for multi-body system analysis and finite element analysis. Therefore, this approach can be helpful for engineering applications to combine, filter, and transform data originating from different domains.

Other database features such as storage management and query processing suit engineering applications well where large amounts of data normally need to be stored and used for efficient retrieval and computation.

2.2.2 Distributed object computing
CORBA is one of the important distributed computing infrastructures to support the integration, sharing, and exchanging of information, especially concerning object technology. It is widely used in manufacturing, software, and telecommunication with companies such as Boeing, LG, Cisco, and Nokia [21]. CORBA invokes methods on distributed objects residing anywhere on a computer network as if they were local objects. CORBA describes functionality of an object through the Interface Definition Language, a host independent and declarative language that has mapping standards to major programming languages such as C, C++, Java, and Lisp. CORBA therefore provides programmers with a transparent way to communicate objects in a heterogeneous, distributed environment without considering hosts, operating systems, and programming languages. In this work, CORBA is used to access, operate, and distribute objects of CAD system.

2.3 The system configuration

An overview of the M-Sync system is shown in Figure 2. The system consists of one database Name Server and any number of geographically distributed CAD systems that are interfaced by local database management systems supporting active rules and an object-oriented data model. The interfaced databases can communicate directly with one another after registering with the Name Server, since the Name Server acts as an agent in the system to provide communication primitives and keep track of the DBMS’s. The Name Server also stores the state of the CAD systems for managing transactions of simple cases.

Design works are performed in the local CAD systems. Instead of only serving as information resources (pure storage), the databases used in this system also assist the design activities during the collaborative design process. More specifically, DBMS’s are responsible for storing CAD data, identifying and propagating data updates to other CAD systems, receiving updates made by other designers via network, consistency control, and transaction management. These functionalities are achieved by using database features such as object-oriented data model, query language, database active rules and functions, and multi-database facilities.
When a designer joins a collaborative design work, their local DBMS peer is automatically registered with the name server. During collaborative design work, the designer works on the local CAD system models, i.e. a replica of the shared model. A designer, for instance Designer 1, issues a “sync” command to the system after finishing modifications on the model and wants to discuss this temporary result with other co-designers. The CAD data is then downloaded to the interfaced DBMS where updates identification and propagation are performed using both database ECA rules and relevant database functions. After being informed about the data updates, the remote designer, e.g. Designer 2, at another location receives the updates and automatically synchronizes the interfaced CAD system. Consequently, co-located designers can immediately observe the modifications made by others, and vice versa.

### 2.3.1 The interface between CAD system and DBMS

The shaded rectangle in Figure 2 illustrates the connection between the CAD system and the DBMS at each peer location, while the arrows indicate the direction of the information flow. The two systems are actually interfaced via the CORBA connection. As described above, CORBA provides a transparent way to communicate objects in a heterogeneous, distributed environment without considering operating systems, so that the DBMS can freely communicate with CAD systems running on different operating systems such as MS Windows and UNIX.
2.3.2 Database schema

Database schema is the structure of a database system that defines how data should be stored and can be defined differently according to different applications. For instance, a native CAD system data structure can be used as the basis to define the schema for data exchange between the same CAD systems running under the same or different operating systems; industry standard, e.g. STEP and CAD Services [22], can be adopted if the data is exchanged between different CAD systems. Issues about STEP-based database schema have been studied in [23-26]. The schema in this work conforms to the CORBA-based native CAD system data representation.

2.3.3 System service modules

The system consists of several modules that provide different services for the system. The configuration of such modules is illustrated in Figure 3, corresponding to the lower part of the shaded rectangle shown in Figure 2.

![Figure 3. System service modules in the M-Sync system](image)

- **Data loader** conforms to the CORBA specification and is used to locally or remotely access and download CORBA-described CAD system data to the database management system. This module is called when the designer issues a “sync” command.

- **ECA rule library** contains a set of database ECA rules responsible for monitoring specific events, such as creation, modification, or deletion on CAD system objects, as defined in the rule declarations. The rules’ corresponding
reactions could be either direct data manipulation on CAD system objects or triggering other modules to perform specific operations, such as identifying updates, communicating databases, consistency control, and transaction management.

- **Update identifier, peer detector, and peer notifier** are three models triggered by specific rules to identify updates and communicate distributed database peers that are interfaced with CAD systems. **Update identifier** performs any update identification to CAD system objects. **Peer detector** recognises the existence and the state (“busy” or “free”) of the distributed DBMSs in the federation. **Peer notifier** collects the updates identified by the **update identifier** and propagates them to the distributed database peers that are known by the **peer detector** and whose state is “free”. The state of a CAD system is stored in the database name server and is set to be “busy” when it starts to synchronise others. Hence, this CAD system does not take any input from other systems during the synchronisation process. This simple transaction management can avoid possible system hangs caused by simultaneous synchronisations that are issued by multiple designers. The above functionalities of the three models are implemented by invoking ECA rules from the **ECA rule library**, and by calling their own database functions.

- **CAD-DBMS synchroniser**. Data consistency must also be maintained between corresponding data in the CAD system and in the database. For instance, in a CAD system, a **part instance** in an assembly is a representation (or a “virtual image” but not a copy) of a **part** used in the assembly. It refers back to the part for its geometry definition, but also contains new properties such as orientation, assembly level and visibility, on top of the geometry of the part being displayed. If a part is deleted or modified in the CAD system, all part instances are automatically deleted or modified. This type of data dependencies is called **referential integrity** in database terminology. If these dependencies can not automatically be transferred to the DBMS through a CAD API, they have to be withheld through some other mechanisms. The DBMS can in itself provide generic mechanisms to support referential integrity. For instance, the DBMS used has such mechanisms that if a type is deleted, all the sub-types of this type and all functions defined on these sub-types are deleted; if a type is removed from an object, all references to the objects as instances of the removed type cease to exist. There are also DBMSs that generically support representation of **part-of** relationships that could be used for modelling the example above. This means that the entity **part instance** could be modelled as a part of the entity **part** and when a part is deleted, all part instances are deleted too. Hence, to solve the referential integrity problem one has to consider what dependencies there are in the CAD system, how they are modelled and how they can be accessed through the API. How should these dependencies be represented in the database and are there any generic mechanism that can be used to facilitate this or do they require special methods. In this case, (1) the CAD system used contains a large number of **part-of** relationships such as part-part instance and assembly-assembly instance,
but integrity maintenance functionalities are not provided for external applications through its API, i.e. in this example the DBMS cannot get a list of part instance objects, which need to be deleted in the DBMS, from the CAD system; (2) although the DBMS used in this work has some built-in referential integrity functionality, it is insufficient in this case. Therefore, in the above example, only object of the part in the database can be deleted, but objects of the part instances of the deleted part still remain. This causes inconsistency between the CAD system and its interfaced database and will cause further errors in the change identification process. For this reason, a mechanism for integrity maintenance must be explicitly modelled in the DBMS to keep data consistent between the CAD system and the DBMS. The CAD-DBMS synchroniser is built for solving this problem, containing relevant dependency knowledge similar to the above example to solve inconsistency problems. Like some of the other modules, the functionality of this model is also implemented by invoking ECA rules from the library and by calling its own database functions.

3. Prototype implementation

The proposed approach has been implemented as a collaborative CAD system environment. In this implementation, the database management system AMOS II [27] is used to interact with a commercial CAD system I-DEAS [28].

AMOS II (Active Mediator Object System) is an object-relational and main memory resident multi-database system. Its core is an open, extensible DBMS supporting an object-oriented data model. AMOS II is also an active DBMS where database ECA, event-action (EA), and condition-action (CA) rule models are supported by AMOS II rule processor. The specified events are updated, added, removed, created, and deleted and are used to monitor updates to AMOS II functions as well as creation and deletion of object instances. AMOS II is furthermore a peer mediator system. Each mediator is an autonomous object-relational DBMS with its own query processor, storage, and catalogue [29]. Every mediator peer can act both as a client and a server to any number of other mediators. The above AMOS II features are directly related to this work and are suitable to represent, store, access, and manipulate complex engineering data from different and distributed data sources as well as to react to specific events.

The CAD system I-DEAS is used to perform collaborative design work. I-DEAS data can be accessed, manipulated, and distributed via its CORBA-based application program interface, Open I-DEAS. Since CORBA is a platform independent standard for object computing, an M-Sync system can be implemented on different operating systems. As we have successfully tested, the M-Sync system can directly interact with the CAD system I-DEAS running at the same or different hosts with the same or different operating systems without requiring any re-coding effort.

A sub-assembly of a steam engine, shown in Figure 4, has been chosen as an object to verify the proposed approach. This sub-assembly is a crank system of the
steam engine, as shown in Figure 5. The function of this mechanism is to transform linear movement of the *cross head* to rotate the *crank shaft* via the *crank web* and the *con-rod*, and eventually drive the *flywheel*.

Figure 4. Solid model of a steam engine

Figure 5. Crank system - a sub-assembly of the steam engine

### 3.1 The database schema

In this implementation, the database schema conforms to the native data representation in Open I-DEAS, i.e. a CORBA-based object-oriented description for I-DEAS data. Figure 6 illustrates an entity-relationship diagram of the subset of Open I-DEAS data representation defined in AMOS II. The AMOS II object-oriented data model naturally represents such complicated object-oriented data structures. The following codes define parts of the diagram in AMOS II.

```plaintext
create type oi_object properties(state charstring, name charstring);
create type 3D_model under oi_object properties (date charstring, version integer, label integer);
create type Part under 3D_model properties (madeOf Material);
```
create type Dimension under oi_object properties (value real);
create function owingPart(dimension) -> part as stored;

oi_object is a root type of any Open I-DEAS object. It has attributes state and name that are strings. Type 3D_model is a subtype of oi_object. Besides its own attributes, viz. data, version, and label, it also inherits the attributes state and name from its supertype oi_object. Similarly, type part is subtype of 3D_model, so that it inherits all the attributes from oi_object and 3D_model. The relationship function owningPart is for linking two types, part and dimension.

3.2 Update identification and propagation

Only updated data at the local workplace needs to be identified and propagated to other locations. Therefore, the necessary information transfer is minimised. The part dimension is chosen as an example to show how database active rules can be used to identify and propagate updates. While data loader downloads CAD system data to the database, the rule updateDimension actively monitors values of any dimension. If the data loader updates the value, the rule sets the state of that dimension to be “MODIFIED”. The rule definition is as shown below.

create rule updateDimension() as
from dimension d
on updated(value(d))
do set state(d) = 'MODIFIED';
After the download procedure is completed, update identifier collects all the updated, created, or deleted data and its related information, and then sends them via the peer notifier to the available peer databases that have been detected by the peer detector. In this example, values and names of the updated dimensions and their part names are collected by the following query.

```sql
select name(p), name(d), value(d)
from part p, dimension d
where state(d)='MODIFIED' and owingPart(d)=p;
```

### 3.3 Consistency control

As described in section 2.3.3, the CAD-DBMS synchroniser is designed for controlling consistency between data in the CAD system and the database. Database functions and ECA rules stating the CAD system dependency policy are built to solve the inconsistency problem caused by the CAD system data dependency. The following code defines the rule and relevant functions to maintain consistency between the database and the CAD while parts are deleted in the CAD and then synchronised with the database. As shown below, functions `deletePartDimensions`, `deletePartInstanceTransforms`, and `deletePartInstances` are created to delete part dimensions, transforms of part instance, and part instances. If the CAD deletes any part, the rule `deletePart` first calls these three functions to perform deletion on the dependent objects of this part, and then deletes the part itself so that the consistency is preserved.

```sql
create function deletePartDimensions(part p)->boolean as
for each dimension d
where d=getallkeydimensions(p)
begin
    delete d;
end;

create function deletePartInstanceTransforms(part p)->boolean as
for each partinstance i, transform t
where p=owingpart(i) and i=owinginstance(t)
begin
    delete t;
end;

create function deletePartInstances(part p)->boolean as
for each partinstance i
where p=owingpart(i)
begin
    delete i;
end;

create rule deletePart() as
from part p
on updated(state(p))
when state(p)='DELETED'
do begin
    deletePartDimensions(p);
    deletePartInstanceTransforms(p);
    deletePartInstances(p);
    delete p;
```
4. Conclusions and future work

This work employs database technology and distributed computing to the area of engineering collaborative design. The focus is to study some of the important technologies and architectures that can facilitate distributed collaborative design using conventional CAD systems. Through the development of the M-Sync prototype system, it is shown that database technology can provide important functionalities that are required in a collaborative design system using conventional CAD systems. In this work, the database management system not only provides direct support for the effective storage of and efficient access to complex CAD data, but also assists collaborative design activities by offering advanced database features such as an extensible object-oriented query language, active rules, and multi-database facilities. Specifically, the DBMS is used to identify and propagate data updates made by the CAD system, which is a critical issue in engineering collaborative design. It further maintains data consistency between the CAD system and the database which is another critical issue when data is distributed and replicated. This work also employs the distributed object computing to access, operate, and distribute CAD system objects between the CAD system and the DBMS in a platform independent manner.

The system architecture and basic transaction management are also studied. The present work extends functionalities of the stand-alone and single-user CAD system and facilitates synchronous collaborative design where distributed designers perform online interactive discussions regarding incremental modifications of a CAD system model. The proposed approach is implemented as a collaborative CAD system environment where distributed design information can be actively synchronised between co-located designers when typical collaborative design tasks such as solid and assembly modelling are being performed.

An important future research issue is to extend the current system functionality to integrate data from different engineering domains. The essential part is to map and integrate high-level domain schemas that are used to define different engineering domains. Here, the STEP standard can be useful since it provides widely-accepted and mature domain schemas for various engineering applications. A database management system that supports a wrapper-mediator approach and a powerful query language further provides means to efficiently access, manipulate, combine, and integrate heterogeneous data. Therefore, it is believed that the present work along with these two complementary technologies will make it possible to accomplish an engineering collaborative design environment where CAD and other engineering software systems can be integrated as plug-ins through standardized interfaces.

Another future research issue is to merge data from different designers or versions. Here, it is believed that mechanism such as database transaction
management and version control can be useful. However, in traditional database applications, a transaction is a short-term atomic unit that either succeeds or fails as a whole. A common approach to maintain the atomicity of transactions is to allow the undoing of a failed transaction by rolling back the database to a previous stage. However, this may be much more complicated in an engineering environment since CAD transactions are normally more long-term, i.e. the designer checks out a model and works on it over an extended period (hours or days), and the design process is typically a probing activity. It progresses by trial and error rather than linearly. Therefore, the management for such transactions should be carefully handled. Open nested transactions such as SAGAs [30] provided by database technology are expected to be helpful to manage such long-term transactions. Furthermore, mechanisms for version control, which is a complex topic in engineering design, must be supported to manage different design versions. Hence, the designers should be able to merge data from different designers and rollback the data in the DBMS and the CAD system to previous stages in case of unsatisfactory design results.

5. Acknowledgment

The grateful financial support was provided by Swedish Agency for Innovation Systems through the Polhem Laboratory at Luleå University of Technology, Sweden. The author also acknowledges Prof Tore Risch and Dr. Kjell Orsborn at Uppsala Database Laboratory for the support of database technology. Finally, thanks to Dr. Peter Jeppsson and his cluster members at the Polhem Laboratory for providing the test solid models for use in this work.

References

27. http://user.it.uu.se/~udbl/amos/
A Database Approach for Information Communication in a Peer-to-Peer Collaborative CAD Environment

Haoxue Ma and Tore Risch

Abstract:
Timely and efficient information communication is a key factor in ensuring successful collaboration in engineering collaborative design. This work proposes a database approach to support information communication between distributed and autonomous CAD systems. It provides the designer an easy and flexible way, a project-based propagation meta-table, to specify what parts of a CAD information model should be communicated to other collaborating designers. A CAD peer manager, containing a peer database that stores information to be exchanged with the other collaborators, wraps each participating CAD system. The peer manager identifies changes made to the CAD model by downloading data from the CAD to the peer database. The changes are detected by stored procedures and active rules in the peer database that are automatically generated based on the propagation meta-table. The updates of the types of data that the user has specified to be shared between peers are timely propagated to other peers via inter-database message passing, thereby minimizing the volume of necessary information to be exchanged. When designers receive information concerning the updates from other peers they can freely incorporate, filter, or delete the received updates by manipulating corresponding messages using a propagation control interface, which is also used to issue user’s commands to download the data from the CAD system to the peer database and lookup the received messages in the peer database. The approach is applicable on any CAD system having a CORBA interface and can be applied also to other kinds of object-oriented interfaces.

Keywords:
Peer-to-Peer systems, peer databases, propagation meta-table, active databases, dynamic rule generation, object-orientation, information communication, propagation control interface

1. Introduction
In distributed engineering collaborative design, successful collaboration heavily depends on timely and efficient product information communication between computer-based tools, such as CAD systems. Collaborative design is normally performed either within the same or between different domains, and current information systems supporting the design activity are typically client-server based.

A peer-to-Peer (P2P) system is a general software architecture that provides an alternative to the traditional client-server architecture. It was initially used for
distributed file sharing and made popular by the Napster system [1]. P2P can be
defined simply as the sharing of computer resources and services by direct exchange
between equal peers rather than clients and servers [2]. It opens a new window for,
amongst others, information communication over the network due to its fundamental
principles, viz. resource sharing, decentralization, and self-organization [3, 4].

Object-oriented methodology is currently used, almost by default, in the field of
engineering information systems, especially in the CAD community. Two important
reasons for this are that it is a natural fit with complicated engineering data
structures and it allows code reusability to make the development of CAD systems
easier. Object-oriented methodology is not only used to develop CAD systems, but
also to define Application Programming Interfaces of CAD systems so that users can
enjoy the benefits of object-oriented analysis, design, and development.

According to the characteristics of P2P computing and collaborative CAD, a few
requirements can be identified and must be met to achieve successful information
communication in such an environment, the aim of the present work:

• Efficient representation of objects having complex structures, operations, and
  methods as found in CAD data representations. This is the basis for any
downstream activities concerning data communication.

• Timely and effective exchange of design ideas between distributed designers.
  This is precisely what “collaboration” concerns. These issues can be further
divided into
  o Identification of updates made in a CAD system
  o Propagation of the identified updates to peer designers
  o Selectable incorporation on updates made by peer designers

• Information communication in a platform independent manner. This is
  required by software and hardware diversity.

• A scalable system architecture. This enables to handle large amounts of
  complex CAD data communicated between many peers without significantly
degrad ing performance.

• System extensibility. It should be easy to extend functionality by including
  new functionality provided by the CAD system.

In this work, a peer-to-peer (P2P) database management system [5] based
approach is proposed to facilitate information communication in a collaborative
CAD environment. Each participating CAD system is distributed and autonomous. It
is wrapped by a peer manager system that contains a peer database storing objects
to be exchanged with other peers. The peer database is managed by an object-
oriented and active peer database management system. For easy extensibility, the
designer simply needs to fill in one project specific propagation meta-table
containing only the names and attributes of CAD entities to be propagated to other
peers. Stored database procedures and active rules [6, 7] in the peer databases are
automatically generated from the propagation meta-table. The identification and
propagation of CAD updates to be shared with other engineers are handled by these
stored procedures and active rules when the peer manager downloads design data
from the CAD system. Inter-database message passing between the peer databases
propagate the CAD updates. Through a propagation control interface, remote
designers can issue commands to download the data from the CAD system to the peer database, lookup the received messages in the peer database, and freely incorporate, filter, or delete the received updates by manipulating corresponding messages. Accepted changes are then uploaded from the peer database to the receiving CAD system.

This approach is implemented on a CORBA [8] platform. It is applicable to any CAD system having a CORBA or other object-oriented interfaces.

2. Related work

Recently, many research frameworks and prototypes have been introduced to facilitate collaborative design with different focuses. Tay and Roy [9] developed a collaborative multimedia–enabled 3D CAD software called CyberCAD. This system was built on Java and Java3D and enables distributed designers to build platform-independent web-enabled virtual products as a team. A designer can view his partner’s window, share his model, and drag it to his local window. In CyberCAD a self-developed communication protocol called controller/viewer is used where users can request and pass the control to each other, but only one user at a time can control the model.

Bidarra et al. [10] presented a client-server-based collaborative framework, called webSPIFF, to support integrated design of parts and assemblies using feature modelling. During modelling, team members can have their own product views, e.g. part design view, part manufacturing view, and assembly design view, and can simultaneously manipulate features. All clients connect to the same server running a multiple-view feature modelling kernel and maintaining the complete product model. Another feature-based approach for distributed collaborative design can be found in the system presented by Lee and Kim [11], but with different emphases.

Ramani et al. [12] proposed a thin-client collaborative system, called CADDAC, for shape conceptualization. It allows the creation and modification on geometry in a collaborative mode, and maintains the shape creation history in the database on the server-side. Only the client who has the control, called master-client, is capable of doing real-time editing on geometry over the network. To keep the client thin, all solid modelling and constraint solving operations are performed at the server-side.

Despite much progress, most current systems employ a centralised model (controller/viewer and client-server) to conduct collaborative design. However, compared to a Peer-to-Peer computing model, there are some disadvantages with the centralised model: Designers have either only shared access or private access to CAD models, meaning that one designer has to wait until updating members release the manipulation control to the model. This model also requires high network bandwidth and is normally complicated to construct. Furthermore, the central server can be heavily burdened while performing collaborative work serving many designers of complex products. For instance, in the webSPIFF case, client-specific views of part design, part manufacturing, and assembly design have to be constantly generated and propagated to all clients. Finally, it is potentially less fault-tolerant since the central host is a single point of possible system-wide failure [13].
Using a database management system in collaborative design has been recognised as an important issue. In the pilot system developed by Han et al. [14], product models are stored in a shared database that is formatted according to the Standard for the Exchange of Product Model Data - STEP. The schema of the database is based on AP203 of STEP. Metadata and ontology concepts are used to improve the search capability of the product database.

Urban et al. [15] presented an Integrated Product Data Environment (IPDE), where a STEP database is used as a repository to share data between engineering design and analysis tools. The IPDE is capable of recognizing the STEP concepts, named Units of Functionality, so that users can extract only the relevant parts of a file from the database instead of checking in or out entire exchange files.

Databases have been employed to store file-based product data in the above systems, where sharing and parallel access to the product data are thus allowed. This approach partially solves the inherent problem of file-based information exchange, i.e. a file is basically for archiving purposes and has no or at least insufficient functionality for online collaborative design regarding access to the CAD data structure and high-level functionality. However, it would be more beneficial to take advantage of advanced database capacities, such as stored procedures, active rules, high-level query languages, and P2P database facilities, so as to assist the product design process rather than using a database server as just as a pure back-end repository. As authors have experienced in the present work, database capacities enable a more efficient development of the application and the application is more flexible, less complicated, and easier to maintain compared with using only conventional programming languages, such as C++ and Java. In our system, P2P facilities in addition provide a decentralized architecture where changes are detected by database procedures and active rules and then propagated between peers.

3. The active Peer-to-Peer approach for data communication

In this section, the system architecture and technical issues of the peer database in our approach are described in detail. The presented database definition code is slightly different from the implementation so as to emphasis our methods rather than code details.

3.1 The system setup

The system configuration is shown in Figure 1. Any number of geographically distributed peer designers is working with a traditional single-user stand-alone CAD system. Each CAD system is autonomous and wrapped by a peer manager that uses an object-oriented active P2P database management system (P2P-DBMS). The databases can communicate directly with one another after registering with a Name Server, where database communication information is stored. The exchange of update information between peer sites is conducted via direct inter-DBMS message passing over the network. Designers perform design tasks at local CAD systems through a CAD GUI and then issue collaboration-related commands through a
propagation control interface to the peer database to exchange design information with other designers.

Figure 1. The system architecture

The database management system used by our peer manager is AMOS II - an object-relational multi-database system. Its core is an open, extensible DBMS supporting an object-relational data model [16]. The basic concepts of the AMOS II data model are types, functions, and objects corresponding to entities, attributes, and instances of the engineering information model in this work. The data model is accessible through AmosQL, an SQL-99 [17] related object oriented query language that is used to define, populate, query, and update the database. AMOS II is furthermore an active DBMS where database ECA, event-action (EA), and condition-action (CA) rule models are supported by the AMOS II rule processor. The specified events can be updated, added, removed, created, and deleted and are used to monitor updates to AMOS II functions as well as creation and deletion of object instances. The active rules monitor the “net effect”, i.e. value changes rather than actions [18]. There are two main interfaces, the callin and the callout interfaces, that AMOS II uses to interface with external systems written in other languages such as C, Lisp, and Java. The external systems call AMOS II through the callin interface while AmosQL functions call external subroutines written in other languages through the callout interface. AMOS II is also a peer database system [19]. Each peer is an autonomous object-relational DBMS with its own query processor, storage, and catalogue. Every peer can act both as a client and a server to any number of other peers [20]. These AMOS II features are directly related to this work and are shown suitable to represent, store, access, and manipulate complex engineering data from different and distributed data sources along with timely reacting to specific events.

CORBA is an object-oriented standard for distributed object computing using a client-server model [8]. It is used to provide object interoperability, i.e. enable communication between objects regardless of host languages and platforms.
CORBA requires objects to be expressed in and accessed through its Interface Definition Language (IDL) and thereby hides the complexity of object source code.

In our approach, the application programming interface of the chosen CAD system conforms to the CORBA standard. As illustrated in Figure 2, CAD data is accessed by the peer manager through the callout interface, the CAD Interface and the CAD’s CORBA interface, and then stored as objects in the peer database through the callin interface. The peer DBMS identifies updates, propagates relevant updates to peers, and incorporates updates from other peers.

![Figure 2. The system data flow](image)

### 3.2 Naming mechanism for CAD objects

In a collaborative environment it is very important that each communicated CAD object has a globally unique name managed and operated in a consistent way. An object identifier (OID) is a locally unique system-generated identifier used to refer to a specific object in the peer database. When a CAD object is created in a peer database, an OID with a unique identification number is automatically assigned to the object. However, such an identifier is unique only in the local peer database, meaning that it cannot be used to identify a CAD object in other peer databases of the distributed collaborative design environment. Therefore, we require that each object must have a universal logical key to identify corresponding objects in different peer databases.

In a CAD system, objects of different or same entities may have the same name in the range of a whole model and can therefore not be used as keys directly. However, object names of a same entity in their direct owner/container are always unique. For instance, different solid parts may happen to use a same name in different projects of a CAD model, but their names must be unique in their individual container – their direct owner. This is the foundation in our naming mechanism. An object key is thereby formed by concatenating entity name and owner name of the object, viz. “entityName_ownerEntityName_ownerObjectName_objectName”. Two examples of object keys are illustrated below.
**Part_BIN_Main_handle**
**PartInstance_ASSEMBLY_PumpAssembly_handle1**

where *Part*, *BIN*, *Main*, and *handle* are names of *PART* entity, *BIN* entity, an object of *BIN* entity that contains the part object, and the part object. Similarly, *PartInstance*, *ASSEMBLY*, *PumpAssembly*, and *handle1* are names of *PART INSTANCE* entity, *ASSEMBLY* entity, an object of *ASSEMBLY* entity that contains the part instance object, and the part instance object. This mechanism avoids name conflicts in the global environment.

### 3.3 The Propagation meta-table and dynamic rule generation

The present work provides designers a convenient way to specify project-based information to be communicated between collaborative CAD systems through the propagation meta-table. The propagation meta-table contains names of entities and attributes to be propagated, as shown in Figure 3.

<table>
<thead>
<tr>
<th>Entity name</th>
<th>Attribute name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part BoundingBox</td>
<td>Value</td>
</tr>
<tr>
<td>Part Material</td>
<td>Name</td>
</tr>
<tr>
<td>Assembly</td>
<td>Instance</td>
</tr>
<tr>
<td>Dimension</td>
<td>Value</td>
</tr>
<tr>
<td>Transform</td>
<td>Value</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

**Figure 3.** An example of propagation meta-table

The propagation meta-table is this simple due to:

- CAD system data is treated in the database on an object level rather than as CAD system native data structures, meaning that the objects are independent logical entities in the peer database.
- The naming mechanism provides enough information to distinguish objects and attributes.

Thus, the mechanisms for identification and propagation of entity updates are simplified and they can be performed by using automatically generated database active rules having a particular template, as illustrated below. Note that *entity*, *attribute*, and *instance* in the CAD information model correspond to *type*, *function*, and *object* in the database model.

```sql
create rule update_typeName_functionName as
    from typeName e
    on updated(functionName(e))
    do propagateFunctionChange(typeName, e, functionName(e));
```

where *typeName* and *functionName* are substituted for corresponding names of each entity and attribute specified in the propagation meta-table. *PropagateFunctionChange* is a stored database procedure that propagates relevant changes of the object *e* to other peers. The rule specifies that if there is any change...
to the value of the given function of the given type, the system propagates relevant changed information to other peers.

When the system is initialised for a collaborative design task, it reads the propagation meta-table and automatically generates a set of active rules based on the above template. Each rule maps to one entity attribute and is responsible for determining its value change. These rules are automatically triggered when the procedure for downloading CAD data to the peer database is completed. This mechanism for dynamic rule generation provides the designer the flexibility to specify project-based information to be propagated at any period in the design process, thereby making the system scalable and easy to maintain.

### 3.4 Update identification

Among three possible updates to an object, viz. modification, creation, and deletion, modifications to function values are identified by database active rules whose template was shown in section 3.3. However, the principle for identifying updates on creation and deletion of objects differs from that for function values modification.

To achieve this, one extra function, called `state`, is introduced to the CAD object definition. The AmosQL code below defines a type `CADObject` in the database, which is the root type of any CAD entity defined in the database schema. This type has functions `name`, and `CADID` that returns the object key.

```sql
create type CADObject;
cREATE function name(CADObject) -> String;
cREATE function CADID(CADObject) -> String key;
cREATE function state(CADObject) -> String;
cREATE function operator(CADObject) -> String;
```

The function `state` is essentially a string\(^1\) telling the database system about the state of an object, which could be “created”, “deleted”, “modified”, or “unchanged”. Each sub-type of this root type therefore inherits this function due to the inheritance feature of the object-oriented data model. By checking `state`, the database knows the state of each object and can then act correspondingly to either notify peer engineers about updates or upload updates made by other peers to the CAD system.

In the database, update identification is divided into two phases, namely identification of object existence and modification of values of object functions, as illustrated in Figure 4. In the example here, objects 1 and 3 are deleted, objects 2, 4, and 5 are modified, and object 6 is created in the CAD system during design work.

---

\(^1\) For efficient representation, the strings can actually be encoded bits associated with each object.
Figure 4. The two phase update identification

Before downloading any objects from the CAD to the peer database, the states of all objects are set to be “deleted”. During the download procedure where the first phase of change detection takes place, the peer manager checks the existence of each CAD object to be populated in the database. If the object does not exist, a new object is created with “created”; otherwise the state of the existing object is set to “unchanged”, meaning that at this moment this object exists, but it still remains to be investigated if its function values have been modified. Objects that have not been touched during the download procedure have been actually deleted in the CAD system. The download stored database procedure is defined below in AmosQL.

```sql
create function theCADObject(String ID, String typeName) -> CADObject o as
begin
    select CADObjectNamed(ID) into o;
    if o = nil then
        /* The object ID does not exist, create one */
        set o = createObject(typeName);
        set CADID(o) = ID;
        set state(o) = "created";
    end
    else
        set state(o) = "unchanged";
    result o;
end;
```

After downloading, the second phase takes place, where the database active rules are triggered to determine function values that have been modified. Finally the change-related information is propagated to other peers. Note that the states in the third table in Figure 4 are same to those in the second table. This is because the function `propagateFunctionChange` in the active rules immediately propagates updates to peers so that the states for those objects do not need to be set, as will be described in section 3.5.

3.5 Message construction and broadcasting

Updates are propagated to peers via inter-database message passing. The messages thereby
• must contain necessary information to tell peer designers all details of the updates
• should be designed not only for designers to view the updates, but also for the peer database systems to execute the updates specified in the messages. The designer should be able to modify the data in the database by reacting on the messages in a convenient way.

Therefore, the format of messages is of great importance.

In the present work, a message is defined in the database as an object of type Message having functions name, sender, receivedTime, and content.

create type Message;
create function name(Message) -> String;
create function sender(Message) -> String;
create function receiveTime(Message) -> Timeval;
create function content(Message) -> String;

The strings stored in the function content are actually a database update commands to be executed at receiving peers. If a designer intends to incorporate updates made by certain peer designers, they can simply select the messages sent by those peer designers and execute them in the receiving peers. For instance, if an object has been deleted, the message content should enable the database to identify the object and perform deletion on this object. As record of collaboration design history, such executable messages could actually have more functions telling the designer the state of the message, for instance, whether the message has been read, whether and when the content of the message has been incorporated, etc.

The function createMessage shown below creates a message object in the database. It is called by function broadcast that sends all peers a message – a command to create a message object in the peers’ databases.

create function createMessage(String msgName, String senderName, String msgContent) -> Message m as
begin
create Message instances m;
set name(m) = msgName;
set sender(m) = senderName;
set content(m) = msgContent;
set receiveTime(m) = now( );
result m;
end;

create function broadcast(String msgContent) -> String peer as
for each Peer p
where p = other_peers( )
begin
send(p, "createMessage(" + msgName( ) + sender( )
+ msgContent + ");" );
end;

The function other_peers detects all other peers in the collaborative design while the function send passes a message to peer p for evaluation, but does not wait
for any reply as typically happens in a P2P environment. The database function 
messageName creates a name for this message and sender gets sender’s name of the 
message.

As described in section 3.4, created and deleted objects will be identified after the 
download procedure, and their states are set to be “created” or “deleted”. The function propagateAll can therefore be called to propagate overall updates to peers.

create function propagateAll( )->Boolean as 
begin 
for each CadObject o where state(o) = "created" 
begin 
broadcast("created_object(" + CADID(o) + name(typeof(o)) 
+ sender( ) + ");" );
end;
for each CadObject o where state(o) = 'deleted' 
begin 
broadcast("deleted_object(" + CADID(o) 
+ sender( ) + ");" );
end;
rulecheck( ); /*propagates function values*/
end;

This function first calls broadcast to create message objects in the peers’ databases 
for the created and deleted objects and then triggers the active rules generated 
automatically based on the propagation meta-table to identify objects whose 
function values have been changed. The reason for this operation sequence is that 
the objects must be created before function value updates can take place.

When active rules are triggered, changes to function values of objects of the 
specified types are automatically detected by the database system, and the function 
propagateFunctionChange below is then called to create message objects in peers’ 
databases for those objects.

create function propagateFunctionChange(String fn, CADObject arg, 
Object value) -> Boolean as 
begin 
broadcast("modified_object( fn, objectID(arg), value, 
Sender( ) );" );
end;

The procedure performed during message broadcasting is illustrated in Figure 5.
Incorporation update at peer side

A collaborative system must be designed for social expectations, e.g. collective cooperation or autonomy. In our approach, peer designers are autonomous. It is totally up to the designer to decide whether or not the updates made by others should be incorporated, whose updates should be incorporated, whose messages should be deleted, etc. After the designer broadcasts the messages, the messages are displayed on the peer designers’ screens and corresponding message objects are populated in their databases.

By reading the messages, peer designers know detailed information of the updates sent by each peer and can then incorporate the desired updates to their database by calling the function `executeMessage`.

```plaintext
create function executeMessage(String sender) -> Boolean as
begin
    for each Message msg
        where sender(msg) = sender
        eval( content(msg) );
end;
```

The database function `sender` gets the sender name of the message and `eval` sends its input parameter, a database command, to AMOS II for evaluation. In this case, it executes the calls to the specified stored database procedures, `created_object`, `deleted_object`, and `modified_object`, sent by `sender`. They are defined as:

```plaintext
create function created_object(String ID, String typeName, String sender) -> CADObject o as
begin
    select CADObjectNamed(ID) into o;
    if o = nil then
        begin
            set o = createObject(typeName);
            set CADID(o) = ID;
        end;
end;
```

Figure 5. The procedure of message broadcasting
set state(o) = "created";
set operator(o) = sender;
result o;
end
else
  error("Conflict! Object " + ID + " already exists!");
end;

The function created_object creates an equivalent object to that which was created by the peer sender in the database. The function first checks the existence of the object to be created. If it does not exist, the function then creates the object, sets its object key, and finally sets the state to “created”. If the object exists, meaning that the peer designer happens to have created an object with the same key, the designer will be informed of this conflict. This mechanism can avoid possible data inconsistency between peers. In this case, peer designers should coordinate their actions by using some conferencing facility and groupware [21], such as video conferencing, shared applications, and 3D CAD model viewer.

create function deleted_object(String ID, String sender)
  -> CADObject o as
begin
  select CADObjectNamed(ID) into o;
  if o = nil then
    error("Conflict! Object " + ID + " does not exists and can NOT be deleted!")
  else
    begin
      set state(o) = 'deleted';
      set operator(o) = sender;
    end
  end;
end;

The function deleted_object sets the state of the object that has been deleted by the peer designer sender. Later on, the function that is responsible for uploading updates to the CAD system will delete the corresponding objects in the CAD system and the database function will then also delete them in the database according to this state.

create function modified_object(String ID, String fn, String val,
                                String sender) -> Boolean as
for each CADObject o
begin
  select CADObjectNamed(ID) into o;
  if o = nil then
    error("Conflict! Object " + ID + " does not exists and can NOT be modified!")
  else
    begin
      eval("set " + fn + " (" + o + ") = " + val + ";");
      set state(o) = "modified";
      set operator(o) = sender;
    end
end;
The function `modified_object` takes key `ID`, function name `fn`, and new value of the modified object `val` as input to set a new function value and then set the state of this object to be “modified”.

Figure 6 illustrates the states of objects stored in the peer designer’s database, both before and after executing messages sent by peer A. Other functions for synchronising the database and the CAD system can then upload these updates to the CAD system according to the states of those objects.

```
<table>
<thead>
<tr>
<th>key</th>
<th>state</th>
<th>name</th>
<th>operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>object1</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object2</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object3</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object4</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object5</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>state</th>
<th>name</th>
<th>operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>object1</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object2</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object3</td>
<td>deleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object4</td>
<td>modified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object5</td>
<td>modified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object6</td>
<td>created</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 6. Object states before and after message execution

4. Implementation

The proposed approach has been implemented in a collaborative design environment where a CAD peer manager containing a peer database management system, AMOS II, is used to wrap a commercial CAD system I-DEAS [22].

The CAD system I-DEAS is used to perform collaborative design work. I-DEAS data is accessed, manipulated, and distributed via its CORBA-based application program interface, Open I-DEAS. Since CORBA is a platform independent standard for object computing, our approach can be directly implemented to the I-DEAS system running on different operating systems as we have tested.

When the system starts,

1. the **CAD interface** automatically establishes connection between the peer database and I-DEAS via the database callin, callout interface and the Open I-DEAS server
2. the peer database then
   a. loads the database functions (described in section 3) used to identify any updates made in CAD, propagate the updates to peers, and incorporate the updates to the database.
   b. loads Open I-DEAS-based schema defined specifically for I-DEAS data manipulation. Functions defined in this schema are responsible for downloading I-DEAS data to the database and uploading the database updates to I-DEAS.
c. registers with a Name Server and sets itself into “listen” mode to
listen to inter-database messages passed on by other peer databases.

The designer can send commands to the database via the propagation control
interface where three options can be currently chosen from:

1. Lookup messages and/or upload updates to CAD
2. Download CAD data and propagate updates
3. Quit

In option (1), the system calls the relevant function to check if there are any un-
serviced messages in the database, as illustrated in Figure 7(a). If there are such
messages, they will be displayed and the designer can make a choice between
incorporating messages sent by certain peer designers to I-DEAS, as illustrated in
Figure 7(b), deleting the messages in the database, or keeping all of them.

In option (2), the system first calls the same functions as used in option (1) to
look up messages and/or upload updates to the CAD. The system will then call
functions to download I-DEAS data to the database. A flowchart of downloading I-
DEAS data is illustrated in Figure 7(c). At the end of this procedure, all CAD
objects whose states are “deleted” are deleted after the updates are identified and
propagated to peers. Finally, the state of all objects needs to be reset to “deleted” so
that the system is ready to download CAD data the next time.

![Flowcharts](image)

**Figure 7.** The flowcharts for (a) message lookup, (b) uploading data to CAD, and
(c) CAD data downloading
A sub-assembly of a steam engine modelled in I-DEAS is selected as the object to verify our approach. The assembly consists of two parts *soleplate* and *box bed* as shown in Figure 8.

![A solid model of a steam engine](image)

**Figure 8.** A solid model of a steam engine

The following examples illustrate how different propagation meta-tables can affect the design result at peer sites and how the designer incorporates updates made by the selected peer designer. We assume that the project aim at this time is to modify existing solid models and adjust the positions of parts in this assembly. Besides other modifications to the model, a designer at the local site increased the diameter of a hole and assembled the *soleplate* on the *box bed*, as shown in Figure 9. The designer can set up the propagation meta-table, perform option (2) to check unserviced messages, and download I-DEAS data to the peer database. The peer database system will automatically identify all updates made in the CAD and then propagate only the desired design results specified in the propagation meta-table; in this case it could be dimension, position or both.
After being informed, designers at peer sites can perform option (1) to incorporate the updates made by the local designer. Figure 10 shows the results at peer sites when the sender’s propagation meta-table contains position, dimension, and both.

Moreover, peer designers can read all messages and are able to incorporate updates received from the chosen peer designers. Figure 11 illustrates this selectable update incorporation. Let us assume peer designers A and B simultaneously perform individual design work based on the model shown in Figure 9(a). After completing the design work, their design results that are shown in Figure 11(a) and (b) are propagated to designer C. It is up to designer C to select the desired updates to be uploaded to his local I-DEAS. Figure 11(c) shows designer C’s result when designer A’s updates are selected and incorporated.
5. Conclusions and future work

A generic approach is proposed to support engineering information communication between distributed and autonomous CAD systems. Each participating CAD system has workplace information stored in an object-oriented peer database management system supporting active rules and message exchanges with database peers. Updates made in the CAD system are automatically identified by the peer database and only updates of the specified types are propagated to other peers using stored database procedures and active database rules, and inter-database message passing. When the system is set up, the stored procedures and active rules are automatically generated based on a simple propagation meta-table specified by designer. Furthermore, remote peer designers are able to flexibly incorporate, filter, or delete updates received from others by using a propagation control interface.

CAD data in the database is treated as independent objects identified by unique keys rendering the communication mechanisms simple and system independent. Together with dynamic rule generation, which makes the system scalable, extensible, and easy to maintain, this approach can be plugged into any CAD systems having CORBA and other types of object-oriented interfaces.

In the approach, a local designer is able to easily specify project-based CAD information to be communicated and the remote site designer can flexibly incorporate, filter, or delete updates received from others by manipulating corresponding messages. The approach can thereby be used in a P2P collaborative design environment where designers and CAD systems are distributed and autonomous.

Furthermore, the approach can be used in either synchronous or asynchronous collaborative design, since peer designers can be timely notified about the updates and the messages recorded in the database can be used not only for designers to view but also to be maintained as project notes and history.

Though the current system can automatically identify and propagate updates of a CAD model, the downloading and uploading of CAD data are still manual, i.e. much effort for writing new methods in conventional programming language are needed when new types of CAD functions are added to the system. It is therefore necessary
to develop high-level mappings of types and functions between CORBA standard and our peer database system to enable automation of CAD data downloading and uploading. Related work, such as in [23], is expected to solve some of these problems.

**References**

5. Special Section on Peer-to-peer-based Data Management, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 7, July 2004


